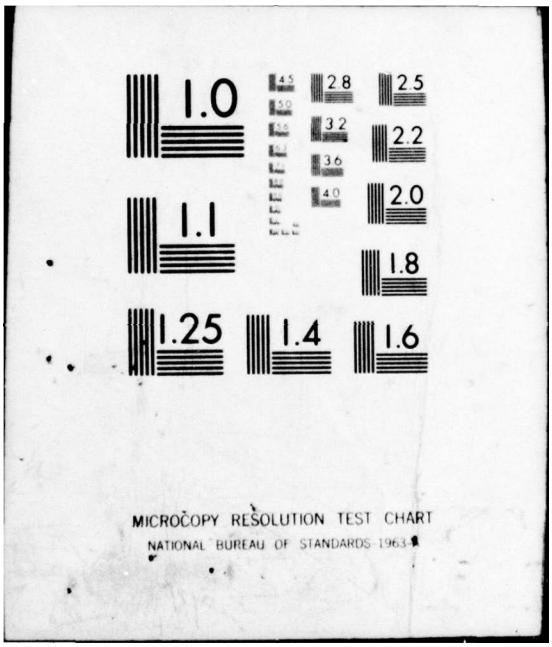
AD-A039 227 NAVAL ELECTRONICS LAB CENTER SAN DIEGO CALIF F/G 9/2
MICROCOMPUTERS AND SYSTEMS DESIGN. ELECTRONIC SYSTEMS CAN HAVE --ETC(U)
FEB 77 W J DEJKA
UNCLASSIFIED NELC/TD-507 NL

1 OF 1
AD A039227

END

DATE FILMED
6-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NELC / TD 507

(12)
B.S.

Technical Document 507

MICROCOMPUTERS AND SYSTEMS DESIGN

Electronic systems can have built-in computational and self test capabilities

William J Dejka

15 February 1977

Prepared for
NAVAL AIR SYSTEMS COMMAND
(NAVAIR 360)
Washington, DC 20360

NAVAL ELECTRONICS LABORATORY CENTER
San Diego, California 92152

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NELC Technical Document 507 (TD 507)
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: NELC/TD-507

4. TITLE *(and Subtitle)*: MICROCOMPUTERS AND SYSTEMS DESIGN. Electronic systems can have built-in computational and self test capabilities.

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): William J. Dejka

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Electronics Laboratory Center, San Diego California 92152

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62762 (element), 54582 (task area)

11. CONTROLLING OFFICE NAME AND ADDRESS: NAVAIR 360

12. REPORT DATE: 15 February 1977

13. NUMBER OF PAGES: 70 (71 pp.)

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*: Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

403 940

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Microelectronics	System architecture
Microcomputers	Software organization
Large scale integration	Application-specific computers
System design	System test and monitoring

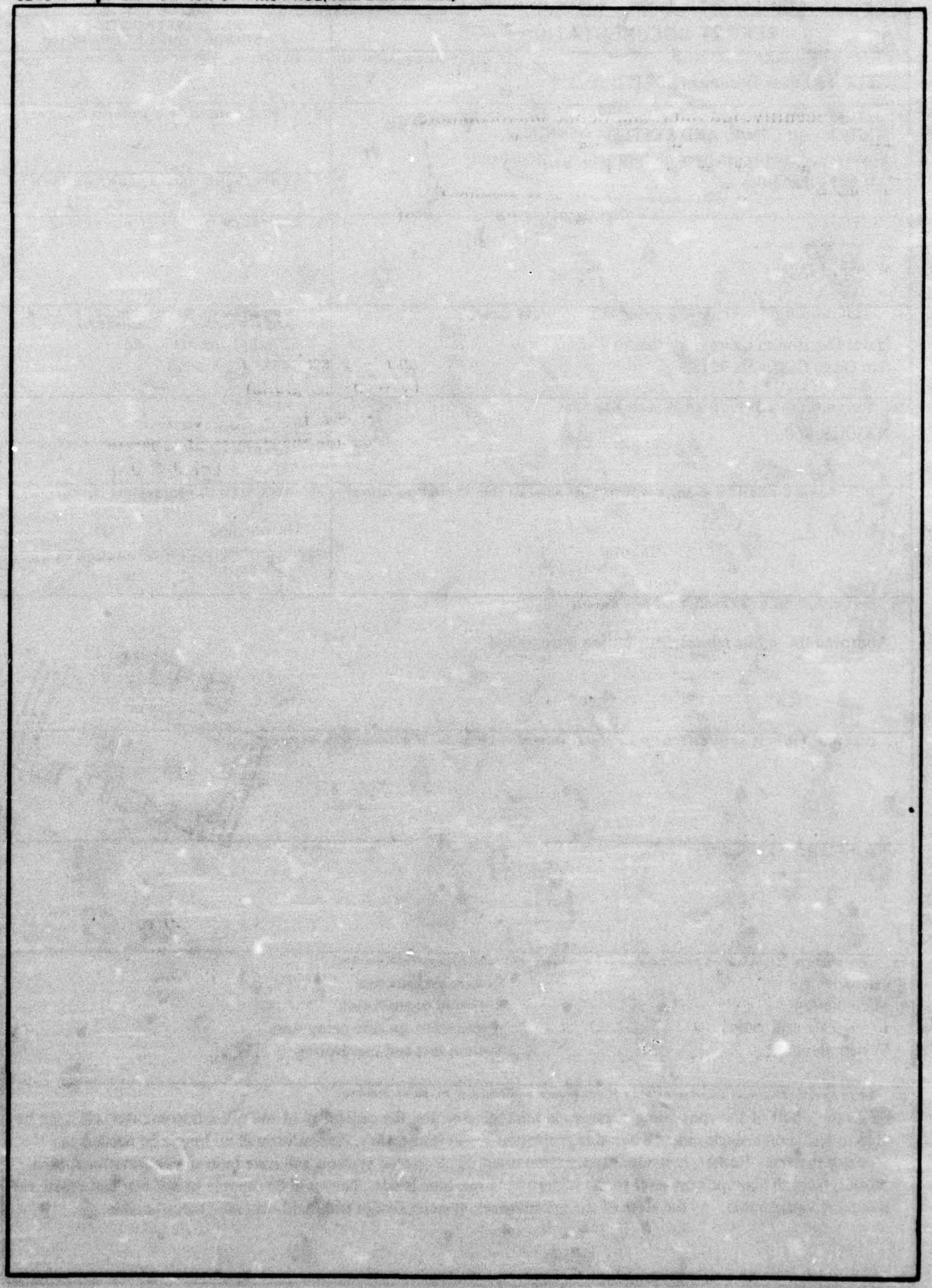
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

As the result of the continuing advances in microelectronics, the capability of today's minicomputer will soon be able to reside on a single chip. When this projection grows into reality, computers will no longer be needed as separate systems. Rather, individual application oriented electronic systems will have built-in computational capabilities, from the component level to the highest man-machine levels. They will be capable of self test and repair and will be reconfigurable. As the state of the art advances, systems design itself will become programmable.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*



OBJECTIVE

Identify, interpret, and define microcomputer uses in system design that will transpire soon and through the 1980s and 1990s. Estimate the impact of these advances on the design of Navy systems. Translate these findings into specific recommendations and applications that will enable the Navy to assume the initiative in specifying future systems designs and configurations, in integrating hardware, software, and firmware, and in implementing self repair and self reconfigurable systems.

RESULTS

1. The state of the art in microelectronics and large scale integration (LSI) is advancing from the integration of circuits to the integration of systems.
2. The microcomputer of the future will be a single chip having the capability of the present minicomputer. This advance heralds the demise of separate computer systems, which will be replaced with application oriented, multicomputer systems designed to have built-in computational capabilities from the component level to the highest man-machine levels.
3. Systems having self test, self repair, and self reconfigurable capabilities are attainable. The impact of these capabilities on Navy mission effectiveness should not be underestimated.
4. Component manufacturers are gaining the ability to generate programmed device designs, versus the general purpose device design approach that has been pursued for years. Such inevitable industrial transitions are already occurring in some segments of the commercial marketplace such as watchmaking.

RECOMMENDATIONS

1. Initiate separate studies to determine the problems, capabilities, and future guidelines to design multi-microcomputer avionics.
2. Explore new concepts of system design in which microcomputers are incorporated. Emphasis should be on fault tolerance, self test and repair, microcomputer primitives, and dedicated system design.
3. Undertake follow-on studies to project and quantify capabilities that may be achieved through the new systems design concepts in flight control, weapons control, and sensor integration.
4. Explore the impact of new component microcomputers on robotics and in artificial intelligence applications. The potential applications of this new technology could be astounding.

ADMINISTRATIVE INFORMATION

Although this task was conducted by Computer Sciences Department (Code 5000) of NELC it reflects the ideas, thoughts, and concepts of many involved in the microcomputer field. NAVAIR 360 was the principal sponsor.

CONTENTS

ILLUSTRATIONS

INTRODUCTION

This report projects advances in design concepts that will result from applying microcomputer technology. Many projections and trend analyses have been performed in the past, and all agree that the microelectronics revolution has only begun. This projection focuses on design approaches and concepts that will be feasible as a result of the advances in large scale integrated circuits and the microcomputer. In particular, this projection indicates that electronics in the 1980s will move from the integration of circuits to the integration of systems.

PURPOSE

The advance of large scale integrated microelectronics will provide the capability to fabricate 100 000 gates on a single substrate in 1980 and 10^6 gates on a single substrate in 1990. This capability will make a variety of new architectures (software organizations) economically feasible. Particular system architectures will focus on regular or iterative arrays of microcomputer building blocks for signal and data processing. These architectures will employ parallel processing to achieve orders of magnitude increases in processing speed. Costs will be reduced by the elimination of a great many types of microcomputer devices and by industrial volume production of microcomputers.

Perhaps the most important advantages of future microcomputer architectures are the simplicity and effectiveness of systems test, monitoring, and control. The structuring of the architectures in simple arrays overlaid by test functions will reduce the complexity of systems. For example, the test and monitoring of a system can be structured and overlaid on the functional and control architectures to achieve independent operation. The simplicity of layered architectures will in turn simplify design and improve performance and system availability.

High system availability has always been a military requirement. The microcomputer will provide a test and monitoring building block with significant capability, yet will be a cost effective tool. The microcomputer will be dedicated to test and monitoring and be designed into the system.

Perhaps the most interesting concept is the use of dedicated microcomputers in functional directed-flow graph architectures. The designer arranges the mathematical algorithms in the form of directed-flow graphs and implements the design by using a dedicated microcomputer to execute each function. In effect, this is a return to analog design approaches in which each device performs a single function, in contrast to the von Neumann computer, in which one multifunction unit performs all functions in sequence and stores results until they are required. Directed-flow graph implementation simplifies design by simplifying control and reducing complexity.

Large general purpose computers are inherently flexible but do not represent the most effective way to solve specific problems. Specific designs based on a large number of microcomputers and tailored to specific applications will be more efficient and economical. In general, the new era of microelectronics will result in reduced use of general purpose computer systems and increased use of application-specific designs using microcomputers.

Required system flexibility will still be obtained through software and programmed control, but the systems will employ more hardware to perform functional processes. This trend is a continuation of the use of hardware to replace software functions such as FFTs.

In summary, the advance of microcomputer technology will result in unique systems design concepts. The purpose of this report is to identify new concepts, ideas, and designs which can provide new systems capabilities. New ideas and concepts will be the basis for a more detailed prediction of specific systems design concepts such as command control, distributed data base management, multidata correlations, etc.

STUDY APPROACH

An overview of the approach to this study is shown in figure 1. The most significant base technologies are

Large scale integrated circuit technology

Microcomputer design technology

Computer system technology

Software technology

Test and monitoring technology

By trending each of these technologies, a 1990 baseline technology is established. This baseline will support new design concepts, which will then be explored. Finally, ideas will be integrated into a projection of systems capabilities, and guidelines and recommendations will be presented for future action.

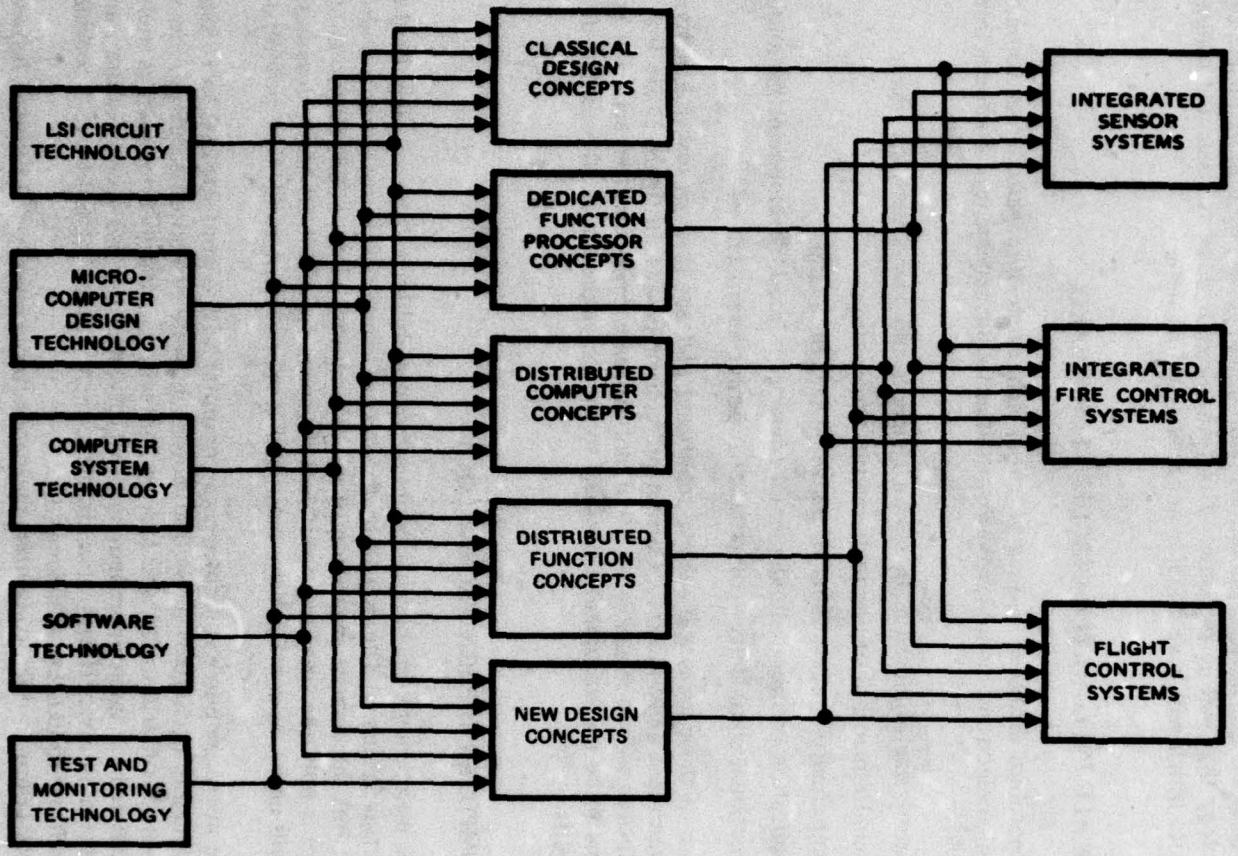


Figure 1. Overview of the study.

TRENDS IN TECHNOLOGY

The projection of trends in new systems design concepts is based on establishing a baseline of new technology which will be available at some time in the future. Today many forecasts on LSI devices, computers, and systems exist, each more or less agreeing on the technology that will be available. This technology forecast is a selected summary of those factors which will have a major impact on system design.

LARGE SCALE INTEGRATED CIRCUIT TECHNOLOGY

Theoretical analyses[1] show a potential eight-order-of-magnitude development increase in microelectronics complexity. Possibilities for advancement of microelectronics devices include

Feature size–dimensions of circuit elements and spaces between them.

Overall dimension of complete circuit–larger chips.

Component count–number of active devices on a chip.

Replication precision–fundamental advances in mask development techniques.

Functional complexity–computational power improvement.

Each of these areas will achieve significant plateaus through research supported by the government and industry. Current projections (shown in figure 2) are that 100 000-gate LSI devices will be available by 1980 and 10^6-gate devices will be achieved by 1990. There seems to be no fundamental obstacle to the development of 10^7- to 10^8-device integrated circuits.

MICROCOMPUTER DESIGN TECHNOLOGY

The microcomputer, because of its programmability, will be a widely used building block in future systems. The impact of the microcomputer on design is now widely recognized, but the current generation of microcomputers (Intel 8080, Motorola 800, AMD 2900) is only the first, and significant advances are possible. Major developments certainly will occur, as some already have, and it is important to identify potential advances.

The microcomputer is defined as a computer with a word size of 2 to 16 bits and a memory ranging from 1000 to 64 000 words. In spite of limited accuracy, low speed for double-precision arithmetic, and lack of hard-wired logic functions such as multiply, microcomputers have microprogramming capability, multiple I-O access features, and some extensive instruction repertoires. Nevertheless, their current capabilities are considered crude in contrast to existing minicomputers.

In describing a microcomputer, the parameters generally fall into the following areas:

Processor

Memory

---

[1] Moore, Gordon E, Progress in Digital Integrated Electronics, International Electron Devices meeting, Washington DC, December 1975

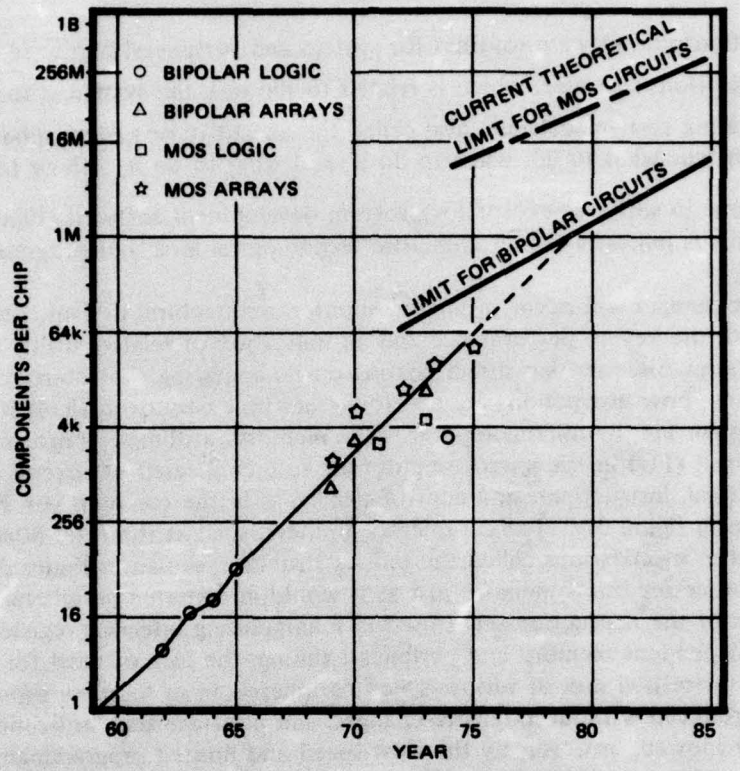


Figure 2. Projection of LSI components per chip.

Input-output

Software

Size, expandability, flexibility.

As a rule, microcomputers are single-address binary machines with 2 or 4 bit bipolar processors, which can be paralleled into large processors or full 8 bit processors without expansion capabilities. Nearly all the processors have typically from one to eight general purpose hardware registers.

Memory is solid state, which poses problems of volatility and comparatively high cost. Memories seem to be the cost driving element in microcomputer systems design. Memory is designed in increments now limited to 64 000 main-memory words. The input-output section is limited to two schemes: processor or program controlled I-O through one or more hardware registers in the CPU; and direct memory access (DMA) with memory cycle stealing but without intervention from the processing unit. The interrupt structure of the microcomputer is integral to the CPU. There are generally two kinds of interrupt structure: single-level no priority, which reserves one storage location while servicing an interrupt; and multilevel systems, which reserve several storage locations as pointers to different interrupt servicing routines. Higher level interrupts can interrupt lower level interrupts.

Software for microcomputers can be differentiated into several classes:

1. Program development software needed by the user to develop his programs for particular applications. This class consists of editors, assemblers, debugging and utility routines, and perhaps a compiler such as PLM-1.

2. Input-output software routines for system and peripheral hardware.
3. Applications software, which is related to the task the system is to perform.
4. Operating system software, also called the executive or system monitor, which tells what to do, when to do it, and what to do it with or to.

Little has been done in software except for program development software, item 1 on the previous page. This is indicative of an immature technological area, but progress is rapidly being made.

Significant changes will occur in microcomputer architectural designs, since the architecture is both the key to performance and an indication of relative utility and power. The microcomputer architecture has direct bearing on its operating characteristics such as speed and efficiency, how instructions are executed, and how easy or difficult it is to construct the program for the microcomputer. The memory, arithmetic processor, control, and input and output (I-O) in the microcomputer are interconnected by several lines which provide paths for data, instructions, and control signals, as in the common von Neumann architecture shown in figure 3A. Typical microcomputers, such as the Intel 8080 and Motorola 6800, have organizations called the unibus structure, shown in figure 3B. In this architecture, the processor treats memory just as it would an external peripheral device, and the full power of the instruction set is used in manipulating interface registers. Advantages are independent memory and peripheral timing, the lack of need for special I-O instructions, a permitted mix of various-speed peripherals so as to allow some new modules to be introduced without extensive changes, and interface standardization. The simplicity is overshadowed, however, by the slow speed and limited programmability for operations needing a large number of memory accesses.

New microcomputer architectures which will improve speed, simplify programming, and permit new design concepts are already under development. Among them are the following general microcomputer characteristics (figure 3C) for future development.

- On-chip memory. The idea of using an on-chip memory for both data and program to increase the effective speed was explored by Wilkes. Each instruction can be fetched from the local (cache) memory, which is loaded from a large, perhaps slower, central memory. Although fast memory technologies are available, they cannot be exploited if transfers to microcomputers require significant delays. The number of such transfers can be reduced by providing an instruction memory on the chip. Similarly, the memory can be utilized for data as well.

In addition, recent application studies such as DPM[2] indicate that a microcomputer with a 4k-byte memory can be effectively applied in both small and large system design (other than "smart" peripherals and I-O). A memory local to the processor is now recognized as necessary on the device. This recognition will result in its occurrence.

- Multibus architectures. Microcomputer configurations should be capable of providing more than one simultaneous transfer path. To implement these multiport/multibus configurations, functional units such as memory, I-O, and CPU will have more than one access port. The control logic within each functional unit can resolve conflicts and simplify control. One important architectural development will be the separation of program and data memories. Other features are the control structures which can be

---

[2]Honeywell, Inc Report AFAL-TR-73-226, All Semiconductor Distributed Aerospace Processor Memory Study, by MD Johnson et al, August 1973

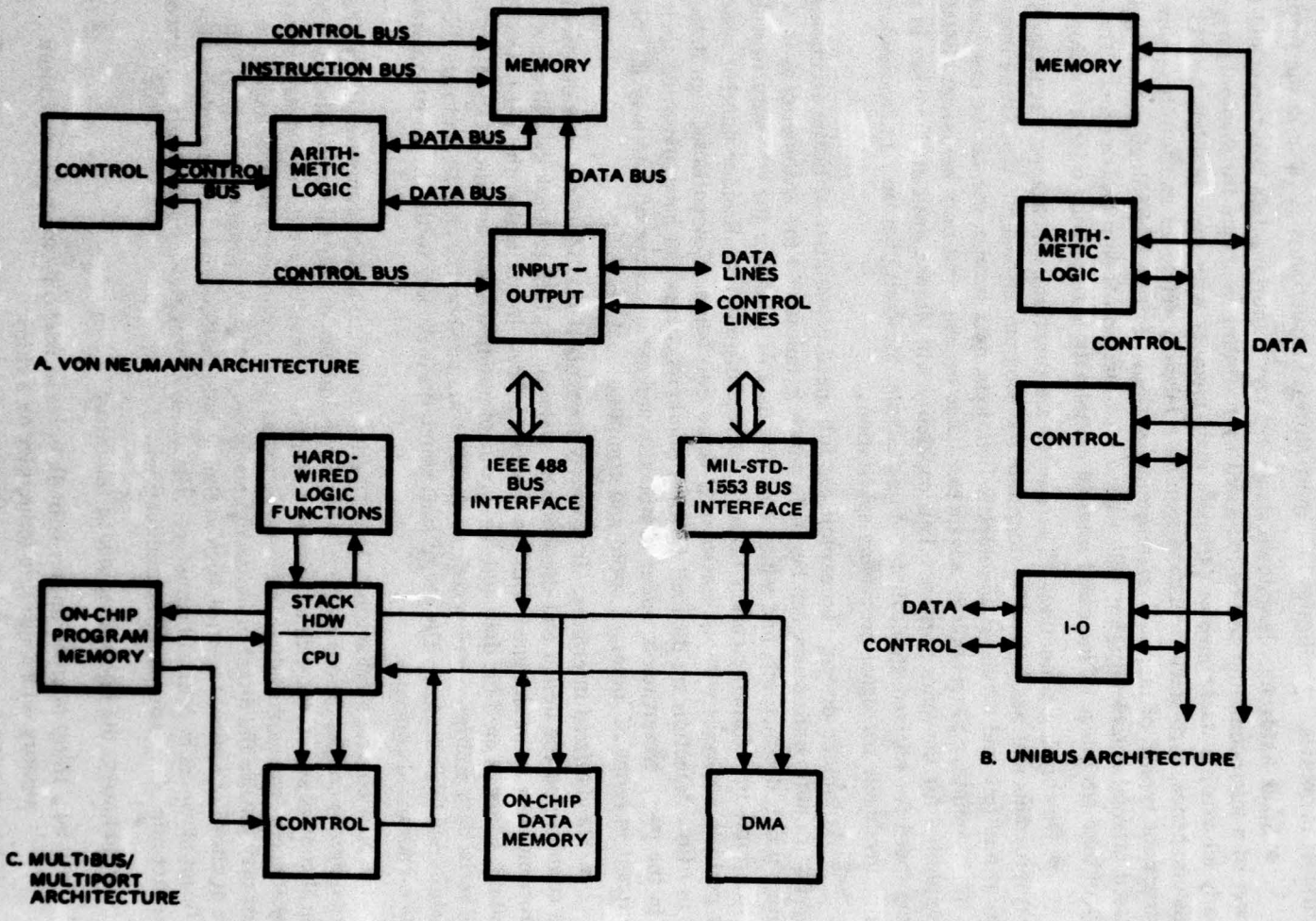


Figure 3. Microcomputer architectures.

developed particularly for interleaving. Figure 3C shows a typical architecture with separate multiport memories. It indicates that multiaddress instruction is possible in the future.

- Stack hardware. Pushdown stacks are very important data structures, and the power of a microcomputer can be increased by providing hardware for manipulating stacks. Nearly all existing stacks involve extensive main memory with stack pointers held in registers; hence, each stack operation requires a memory reference and results in slow stack instructions typical of memory reference instructions. Future technology will allow increased internal complexity, but will place a premium on off-chip transactions. The natural way to take advantage of increased internal complexity is to provide a large on-chip stack.

- On-chip logic functions. There often are functions which appear repeatedly in many programs and which tend to dominate the program throughput (speed). Functions such as multiply and divide, trigonometric functions, data format, etc can be designed into the microcomputer as technology advances. Large on-chip read only memories (ROMs) can be exploited for the same purpose. This capability will allow significant increases in processing speed for selected applications. For example, the multiplier on a microcomputer chip is invaluable for signal processing applications.

- Arithmetic design. To exploit the full speed capabilities of future technology, unique arithmetic unit design can be made which is tailored to the arithmetic, as in floating point, decimal, etc, and which can eliminate or minimize the necessity to wait for the ripple of arithmetic carries through the microcomputer. Residue number systems that perform additions without carries may come into being. Comparisons, sign detection, and overflow detection are difficult in these number systems and have prevented their use in the past. Nevertheless, adder complexity and extra storage of data may not be significant in terms of increased speed and computational capability.

- Standardized interfaces. It is widely recognized that interface considerations have complicated the design and operation of systems. Future technology will allow microcomputers to be designed with standard interfaces. In particular, the IEEE 488 Bus Standard, which is an 8-bit data and 8-bit control busing scheme, and the DOD MIL-STD-1523 serial data multiplex busing scheme could be designed on the chip to eliminate the need for additional logic devices. The use of such standards is the key to the design of systems based on advanced concepts.

- New control mechanisms. The effective control of a microcomputer in distributed multicomputer configurations requires that new techniques of control be introduced. The classical control structures such as priority interrupts are inadequate for new design concepts. Some concepts can exploit direct memory access as a microcomputer control technique. Others may involve the direct or indirect external control of the program counter. Still other methods of external control, such as timing and clock control, can be exploited for significant advances in control techniques. The needs in this area are less clearly understood than other microcomputer architectural features.

In summary, the microcomputer of the future

Will be a 16-bit microcomputer with 4k bytes of read only memory or random access memory on a single ship (computer on a chip).

Will contain logic functions such as multiply, trig, on-the-chip table look-up.

Will contain an arithmetic unit with unique programmability features—stack, hardware, multiaddress design, optimized arithmetic carries.

Will include new external control mechanisms such as program counter control, direct memory access, and timing and clock control.

Will include standard (accepted) interface designs with unique failure design features.

COMPUTER SYSTEM TECHNOLOGY

System designs of the past generally consisted of a basic computer in a specific system organization which could achieve specific performance or cost effectiveness advantages. It was an almost impossible task, however, to evaluate comparatively all the conditions and assumptions used to achieve a particular capability. This is a question not of the veracity of the information presented, but of its comprehensiveness only, in the light of systems complexity. In the past, computer systems were simple uniprocessor architectures which allowed relatively straightforward design, and, of course, straightforward development of technology. Because of the complexities of additional processors and the strong interactions with the characteristics of the workload, it is difficult to ascertain specific trends in the advance of computer systems. Some trends are discussed below, but it should be recognized that others could have been selected.

There is a plethora of computer systems organizations. It is recognized that a system architecture of the past is the computer of today. Some of the computer architectures are shown in figure 4. Among them are the uniprocessor, pipeline processor, array processor, associative processor and other general forms of multiprocessors. Each has specific characteristics which can be exploited in particular processing tasks. For example, the associative processor is designed for data correlation and data file search applications. Each of these organizations was developed and applied in several areas and with significant results.

However, the advance of LSI technology has allowed greater use of hardware for processing and a general reduction of size, weight, and power as in minicomputers. This advance has opened the investigation of computer networks and, in particular, the study and application of multiprocessors and parallel processors, giving emphasis to

Special purpose processors embedded in systems to handle I-O, data communication, and display generation.

Concurrency, particularly with pipeline organizations.

Replication of subsystems or lower level functional units; the use of modular design to increase system reconfiguration flexibility and availability.

Hierarchical memories employed to reduce the time in data transfers.

Input-output devices and displays improved in capability and reliability and more effectively used in designs.

The resultant impact of the advance in technology was the design of systems through multibus, multiprogram computer structures to achieve fail-soft reliable systems. Some current multiprocessor designs are discussed in appendices A and B.

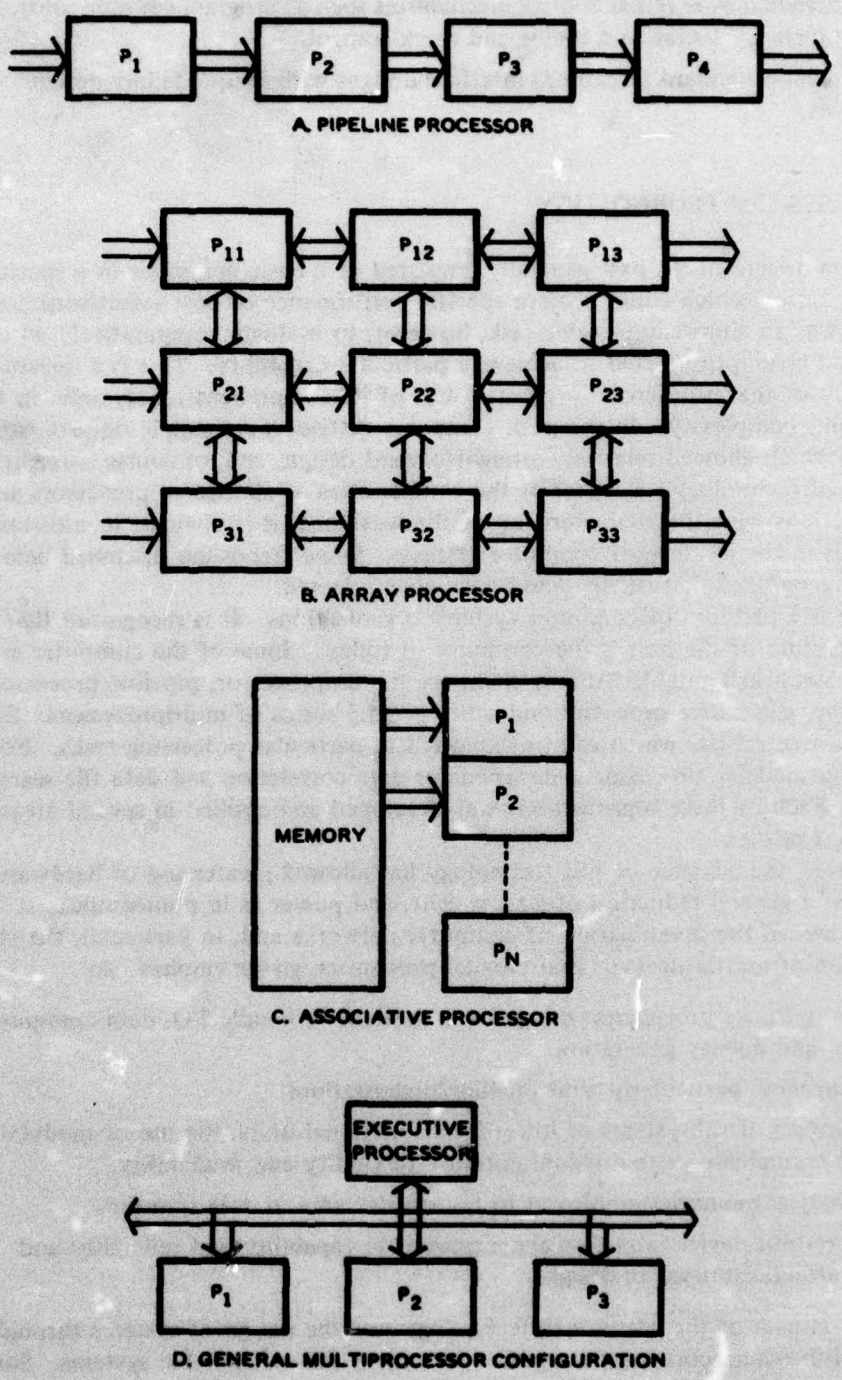


Figure 4. General processing architectures.

SOFTWARE TECHNOLOGY

The hierarchy of software development is shown pictorially in figure 5. The move has always been toward high order languages, primarily to remove the labor burden of the programmer. But the developments have not been entirely scientific, and it is only recently that terms such as structured design, software engineering, and software physics have evolved. The real thrust of software developments has now moved toward making software and its development a scientifically based technology.

Software has come to assume a place of at least equal importance with design and manufacture of hardware. It is not possible currently to design a system without at least designing the major features of the accompanying software package. In fact, it has recently become apparent that it is entirely possible to paper-design hardware configurations which would be extremely powerful but which would be very difficult if not impossible to program efficiently. In general, there is agreement that software has lagged behind hardware design and development.

The following brief definitions describe some important software concepts. A complete survey of contemporary software techniques would be voluminous, but the principal areas listed below identify a hierarchy of technological problems. Each has been the principal element in various research and development efforts.

Executive (or executive routing). A system master program that serves to supervise an entire system. Typically, a real-time system will require an elaborate executive, which may have any of at least the following duties:

Scheduling

Facilities allocation

Real-time control (processing of interrupts, etc)

Data buffering

Execution of diagnostic procedures

Start-restart capability

System utilization and performance data gathering and recording

Development of executives is still an art, and little has been done in the area of executive methodology.

Monitor. A set of system control programs, sometimes under executive supervision, which manages less advanced system operations in detail. Usually the term "monitor" is restricted to "batch" systems in which selection of successive programs to be executed, input-output devices to be utilized, and steps to be taken in the event of difficulties of various kinds constitute monitor principal activities. Advanced features (real-time control, dynamic scheduling, processing of interrupts, etc) are lacking.

Assembler. A conversion program that accepts a machine instruction sequence program written in a mixed symbolic-numerical language, and that produces a working, numerical program in actual basic instruction code. Assemblers were among the first programming aids devised.

Compiler. A conversion program that accepts a program written in an advanced user oriented language such as ALGOL or FORTRAN and that converts such a program into machine instruction sequences for immediate or subsequent execution. Much of the routine programming is done today in languages which are converted by compilers. Accordingly, both the speed of compilation and the efficiency of resulting machine code are

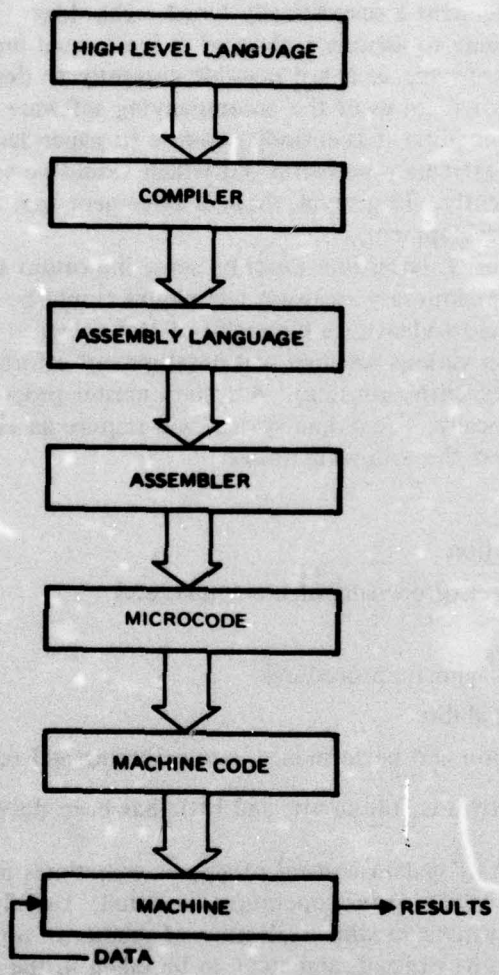


Figure 5. Hierarchical language levels (software processes).

of great importance. These are functions of the source language itself, the nature of the machine for which the compiler was prepared ("target" or "object"), and the competence and philosophy of the compiler designers. The DoD-1 standard HOL effort will result in several compilers to support tactical computing requirements.

List processing language. A set of possible statements, each of which is designed to perform some special task in connection with manipulation (formation, deletions, insertions, etc) of interconnected lists of symbols.

Query language. A set of possible statements involving declarative or interrogatory descriptions of stored information, the extraction of which is sought by the user. Example (in the language used in the retrieval system called "Baseball"): "What were Detroit's winning margins at home before June 1954?" This question appears to be phrased in entirely normal English, but is, in fact, produced under certain rather rigid constraints. Most query languages are rather artificial in appearance. English-like languages are rare, but are the focus of several research programs.

Debugging aids. All of the diagnostic routines and automatic or user optional printouts of contents of portions of memory, etc which help the programmer to decide where his program is in error, whether it is accomplishing what he intended to accomplish, and whether he should alter his basic strategy.

The detailed survey of software technology included in the references may be consulted for additional specific information.[3]

TEST AND MONITORING TECHNOLOGY

The desire to achieve fault tolerant systems, ie, systems that can recover from a permanent or temporary fault, has driven every designer to attempt various implementations of built-in test, software error checking, and system testing. But the full potential has never been reached. Two reasons can account for this failure. First, the technology has not developed fully, except in a few isolated cases. Second, the component and device technology has not advanced sufficiently to permit the cost effective design of fault tolerant, self test, and self repair systems.

Among the areas considered fundamental to fault tolerant and self reconfigurable systems are the following:

- Passive failure masking through hardware approaches

  Redundancy, either at the circuit or subsystem level or with serial-parallel (eg, quad) component aggregates replacing individual components.

  Distributed encoding-decoding—the use of codes in and among registers (includes such techniques as modular arithmetic use of parity and checking).

- Self repair through

  Automatic or semiautomatic switching-out of failed units at any of several possible levels and switching of reserve spares.

  Reorganization of systems by optimum use of nonfailed remaining units.

---

[3] Rand Corporation Report R-1012-PR, Air Force Command Control Information Processing in the 1980's: Trends in Software Technology, June 1974

- Software techniques

  Self diagnostics, self checking and cross-checking.

  Iteration of calculations.

  Execution of complementary check calculations (multiplication after division, etc).

  Execution of calculations using varied sequences of machine instructions, different registers, or both.

  Automatic restriction of computive load to those arithmetic units of a multiprocessing complex which are functioning properly.

In practice, a number of techniques will be used jointly in a given system.

Some recent important and relevant work illustrates what has been accomplished. First, there is the general question of the level at which redundancy should be applied. In the past, redundancy at the gate or component level has not been cost-effective; however, the component is now a computer, and that fact elevates redundancy to what was the subsystem level. At that level, it does appear cost effective. Second, there is the consideration of the "voter," which always must be more reliable than the redundant components. If the redundant component is a computer, however, there is no need for a voter. This alone will be a major step forward in system design.

In using distributed encoding-decoding techniques, numbers are encoded with codes that retain their unique and easily recognized properties when valid arithmetic operations are executed. These number codes are referred to as "residues," and they allow self checking and cross-checking. Parity codes are special cases. As hardware becomes more computationally powerful, these methods are expected to be exploited.

Self repair has been investigated, but complexity, cost, weight, and power have been the limiters of major developments. With low cost hardware, this area will begin to evolve rapidly. A major limitation to our ability to achieve self test and self repair has been the methodology of design. The limitations include calculating fault coverage, percent detect, and other quantitative design parameters.

Software techniques have been exploited, but the need exists to integrate hardware and software. A theoretical foundation for system fault tolerance and self reconfiguration is yet to be developed.

SYSTEM ARCHITECTURE AND DESIGN APPROACHES

The classes of applications dictate different levels of system design approaches of different complexities. To achieve some regularity in the projection of system architecture and design approaches, the following categories are established:

Classical computer system design involving one to three computers.

Multiprocessor systems design involving 1000 to 10 000 microcomputers.

Dedicated multi-microcomputer systems design (100 to 10 000 microcomputers).

Brainlike processor design (one million or more microcomputers).

Each of these categories involves different design concepts and techniques.

CLASSICAL COMPUTER SYSTEMS DESIGN

The basic foundations of computing will continue to be pursued in the classical way—by the application of microcomputers designed to process sequentially organized tasks. This has been the basis of nearly all computation done today, and is supported by the mass of computer science technology already developed.

An example of classical design approach is given by the automation of a process control application, such as depicted in figure 6. It represents a computerized system (in this case, with a microcomputer functioning as the central computer) performing the set-point measuring, parameter control, data recording, and alarm and status monitoring of a large propulsion unit. In effect, the design procedure is the familiar one used in computer system design, by which hundreds and thousands have been completed. The principal difference in the design of computer systems of the future is the size, weight, and cost of the hardware. Not much change in the design procedure is projected. It is possible that when one-chip microcomputers are available, even this design procedure will be modified by the use of many microcomputers to achieve flexibility and reliability.

MULTIPROCESSOR SYSTEMS DESIGN CONCEPTS

The use of many microcomputers in multiprocessor configurations for general purpose processing represents a natural extension of classical computer technology. Systems are assemblages of microcomputers and buses, in various architectures, with primary system control in an executive which is responsible for assignment of tasks, control of data flow, fault monitoring, and other control functions. Many examples of multi-microcomputer multiprocessors have already been undertaken. Some of them were discussed in the previous section.

Principally, the multiprocessor will be applied to problems of broad scope such as the design of a system which can compile and execute ALGOL or COBOL. One such design, shown in figure 7, is the result of research at the University of Illinois.[4] It represents an attempt to define a multiprocessor configuration which is optimized for a class of computational problems. Other such processors can and will be identified and developed, primarily because of the ease of software development (again, exploiting hardware) and the introduction of concurrency.

In the processor (fig 7) memory input-output was optimized through smart memory controllers and processing was optimized through a crossbar switch. COBOL, a business language, involves the processing and handling of large amounts of data, files, etc, as well as performing simple, routine calculations. The architecture is the result of a top down analysis of business/COBOL operations and is potentially a high throughput system.

The system is a multiprocessor requiring a complicated executive, but the use of a separate microcomputer controller will help identify the control functions and separate them from the functional processing.

---

[4] University of Illinois Report UIUCDCS-R-74-638, Program Speed Through Concurrent Record Processing, prepared for the National Science Foundation by RE Strebendt, October 1974

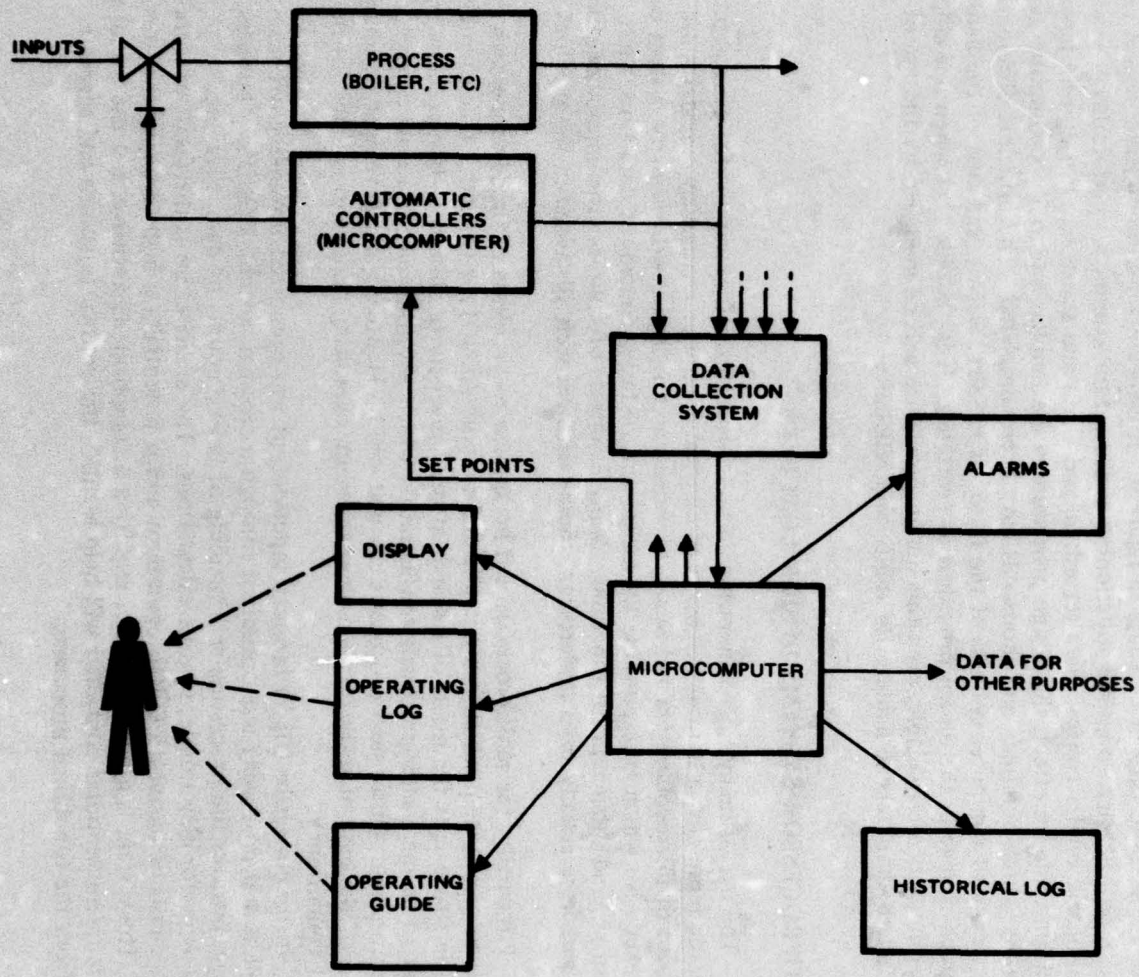


Figure 6. Propulsion computer system.

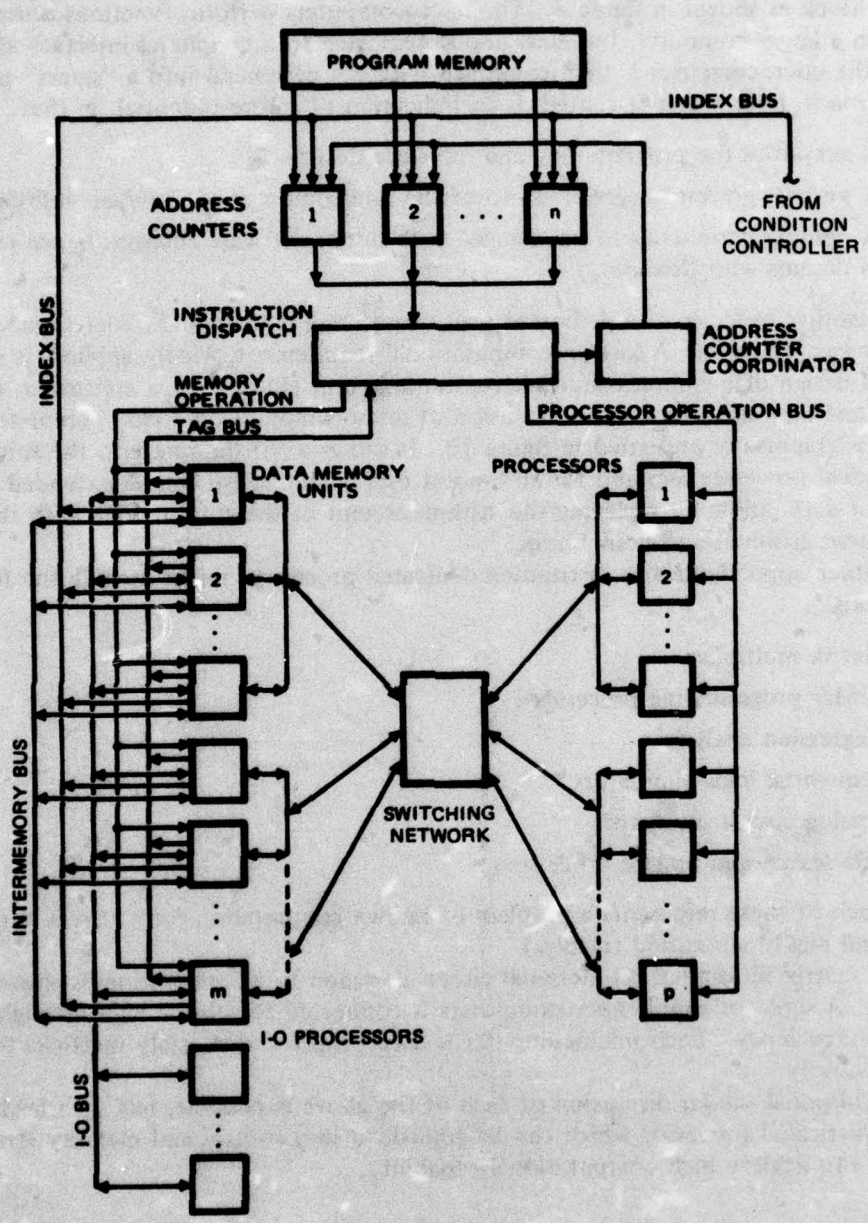


Figure 7. A COBOL machine.

DEDICATED MULTI-MICROCOMPUTER DESIGN CONCEPTS

A recent trend saw the microcomputer introduced as a dedicated distributed function building block as shown in figure 8. The microcomputers perform functions normally performed in a larger computer, but each one is dedicated to a peripheral interface algorithm. In fact, the microcomputer is then combined with the peripheral into a "smart" peripheral. This approach, now widely accepted, is an indication of future potential, in that

It simplifies the programming and interface design.

It permits graceful degradation (over the minicomputer I-O handler approach).

It permits peripherals to be changed with minor software changes, hence results in designs with flexibility.

Another example of a dedicated processor design is the fast Fourier transform (FFT) processor (see figure 9). A known computational requirement, widely applied, is met by a dedicated design of a complex multiplier-arithmetic unit (Butterfly), a controller, and program and data memories. The extension of microcomputer on a chip complete with memory is graphically presented in figure 10. It shows a partitioning into the fundamental mathematical processes required for an 8-point FFT. The design can be extended to any number of data points by replacing the arithmetic unit of the current FFT with the more sophisticated arithmetic (8-point) unit.

Other opportunities in distributed dedicated processors might include the following candidates:

Matrix multiplier

Linear programming processor

Regression analysis

Sequential logic simulators

Analog circuit simulators

File search and update processors

Each of these represents a problem of known computation form that is widely applied and highly structured (regular).

A matrix multiplier, a functional circuit common to all computing, is shown in figure 11. A series of simple microcomputers is connected to achieve high throughput through concurrency. Each microcomputer is used simply and possibly inefficiently, but not ineffectively.

Additional similar discussion of each of the above is possible, but each leads to a more sophisticated processor which can be embedded in a control and memory structure (figure 12) to achieve high computational capability.

DEDICATED ALGORITHMIC DESIGN

The potential of dedicating system functions to microcomputers, as in the directed-flow graph representation of the past, is presented in the hypothetical combat direction system of figure 13.

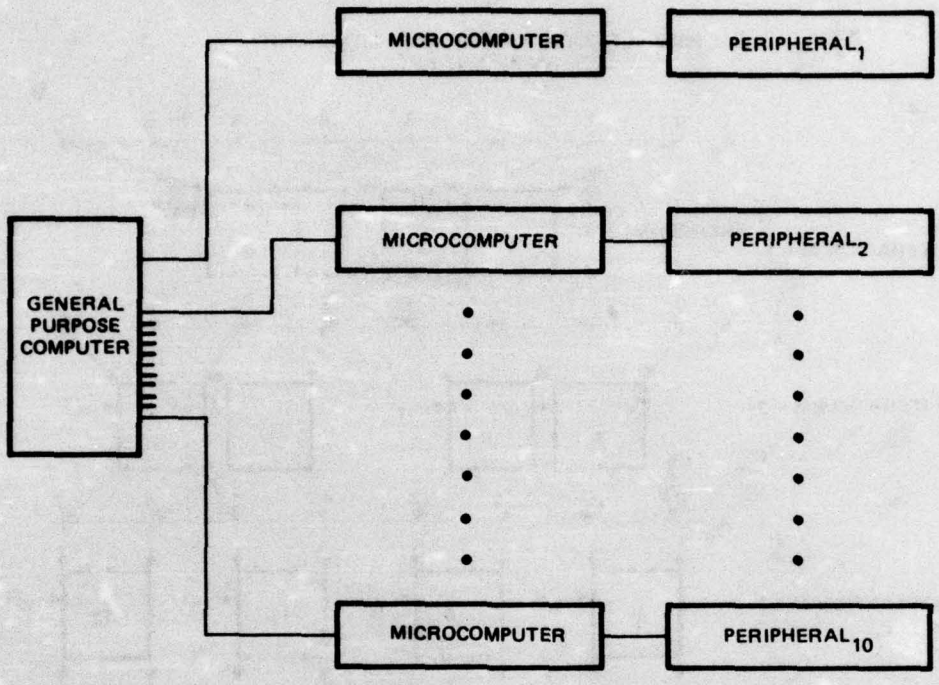


Figure 8. Dedicated distributed multi-microcomputer system.

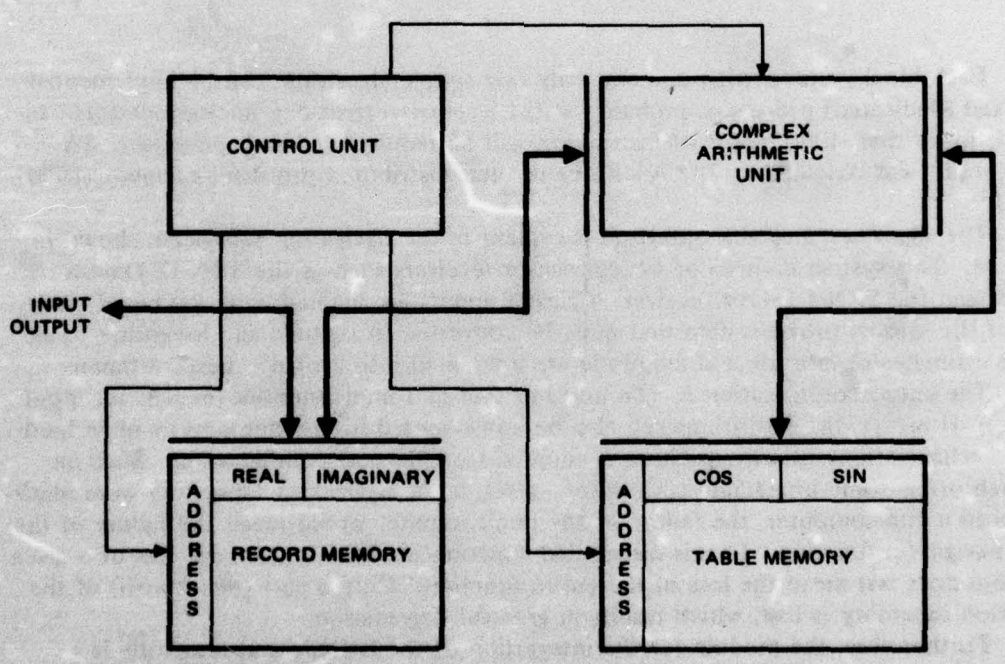


Figure 9. Basic FFT building blocks.

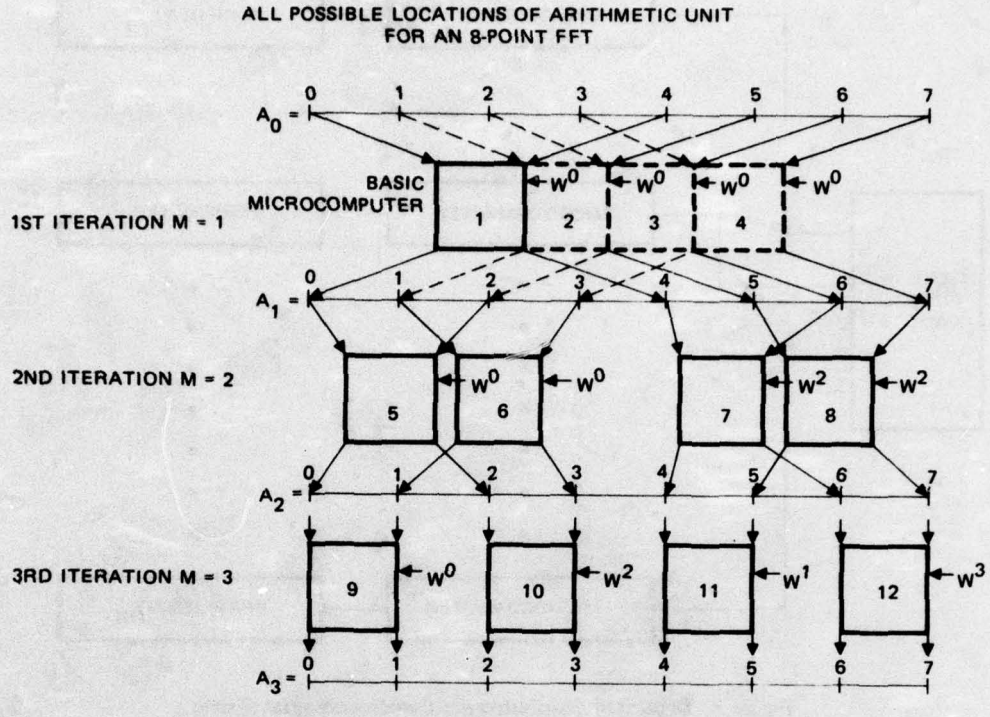


Figure 10. Fully distributed FFT design.

Each block, representing one and only one simple algorithm, can be implemented as a fixed (dedicated) processor, probably at the level of current-day microprocessors. In general, fewer than 4000 words of instruction will be required for each processor. (A similar result was obtained by the Air Force in their distributed processor memory (DPM) study.)

The significance of this approach is evident in the navigation subsystem shown in figure 14. This system consists of various sensor receivers such as the SRN-12 Omega receiver and the SRN-4 transit receiver. Celestial inputs are manual, with keyboard entry. Each of the sensors provides data that must be converted to latitude and longitude. The various estimates of latitude and longitude are integrated into a single "best" estimate.

The entire configuration can be implemented in a minicomputer (except for input sensors). However, the algorithms can also be implemented in microprocessors or in hard-wired mechanizations. Each algorithm is equated to a physical configuration. Such an approach offers some immediate advantages. First, if the navigation subsystem were implemented in a minicomputer, the failure of the minicomputer would mean the failure of the entire navigation function. In this distributed function implementation, the loss of a given algorithm does not mean the loss of the entire function. Only a part (one-fourth) of the navigation capability is lost, which results in graceful degradation.

Furthermore, the module for the integration of the latitude and longitude is a critical nodal point. Its failure would mean the failure of the entire navigation function. Hence, this is a place to insert on-line redundancy. The function is duplicated, and output is used from the other duplicate functional module in the event of failure. This on-line redundancy is cost effective, since it duplicates only a few system cards. To

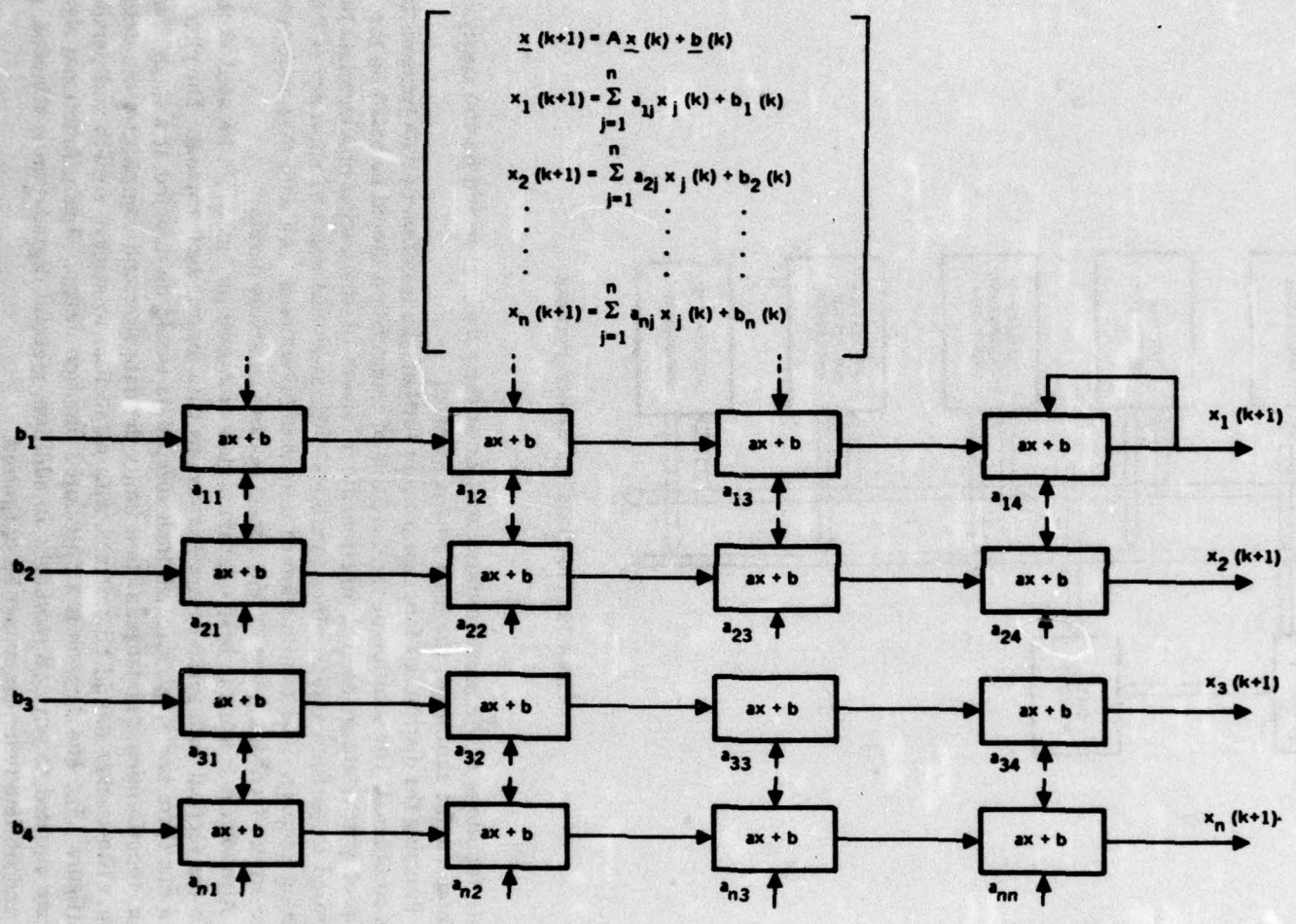


Figure 11. Distributed matrix multiplier design.

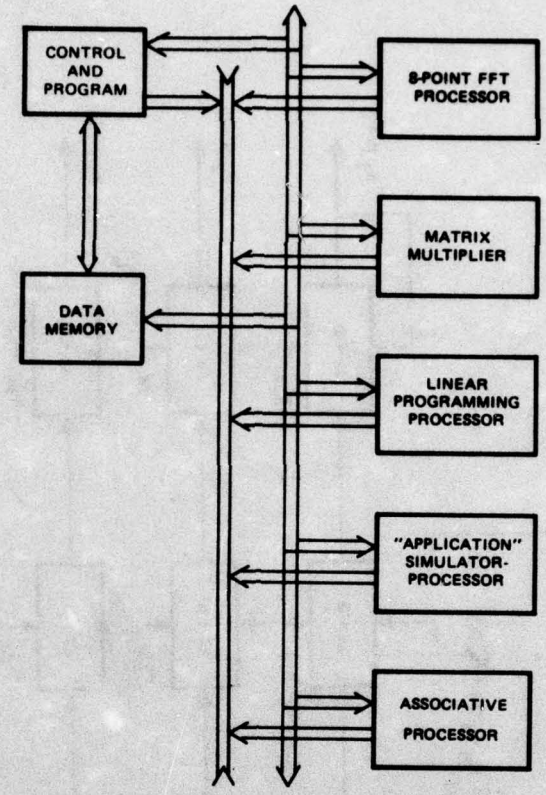


Figure 12. Functionally dedicated processor.

duplicate functions at the minicomputer level or at the gate level would be too costly. At the printed circuit card level, the least cost is involved.

Pursuing the distributed function concept further, the selection of data transmission buses is considered. The solution of various paths of transmission should be based on the locations at which data are needed functionally. In figure 13, the tracker provides data that are required to perform several other functions, such as closest point of approach (CPA) and station keeping. The transmission of data is unidirectional. All data flow is forward; ie, there are no feedback loops except for an occasional status datum.

Although it is not the best solution, a data transmission bus could be used at the input to the tactical data system from the various data sensors such as radar. The plan to put a bus there is limited, since all high speed data would be directed at a single tracker function, thus forming a nodal point that limits the data flow and complicates the design problem. The tracker should be designed with parallelism to achieve growth and parallel inputs (figure 13). The diagram is a distributed function design. Again, functional algorithms are equated to physical modules. In addition, graceful degradation is achieved, and on-line and off-line redundancy can be optimized.

The distributed function approach, in which the functional algorithm is equated to the physical module, represents a technique which clearly identifies the levels of modularity and effectively selects system architecture. Furthermore, it introduces redundancy, self repair, and built-in test—and this feature so significantly influences maintenance, training, and operation cost that it alone deserves separate treatment.

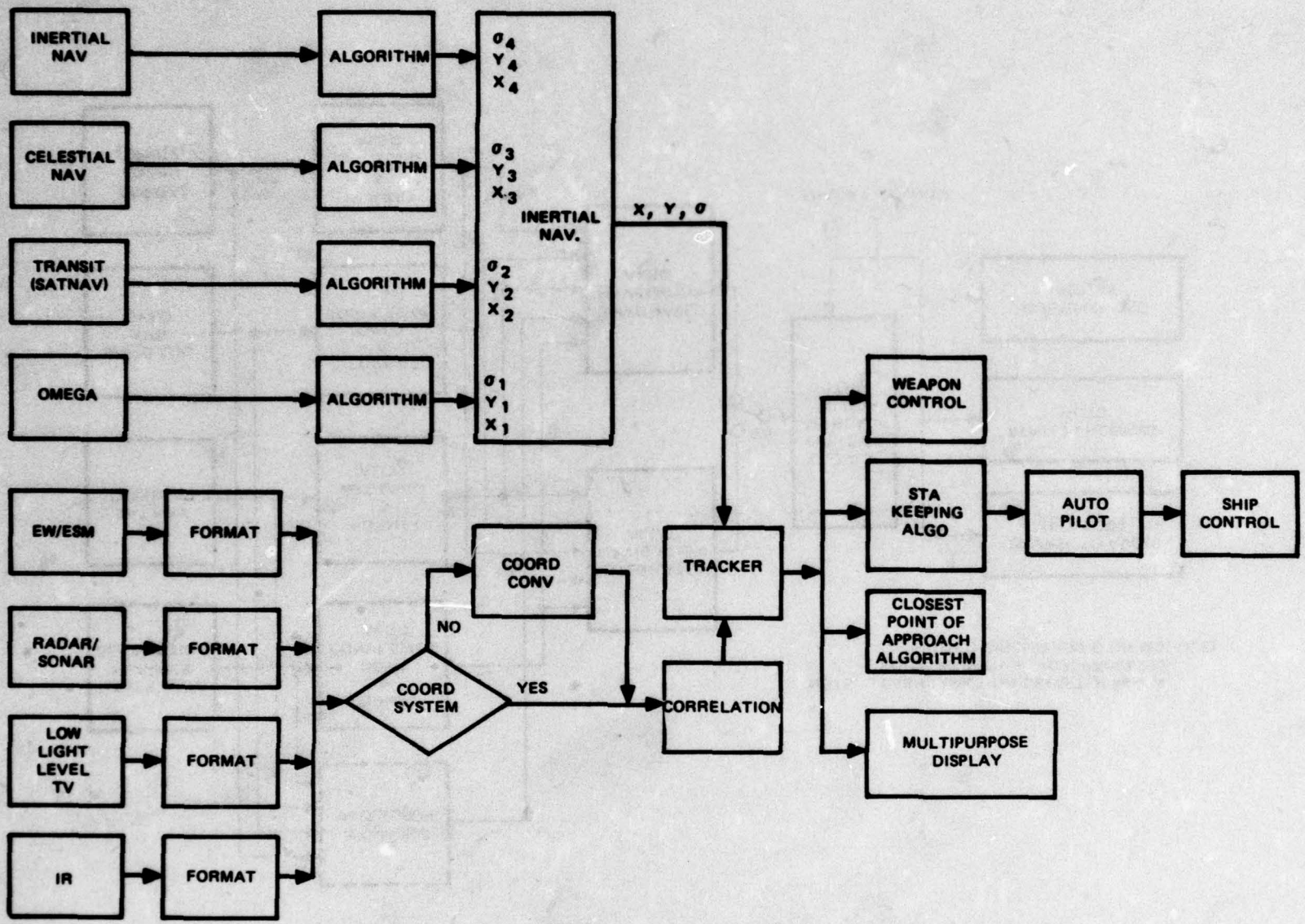


Figure 13. Hypothetical combat direction system architecture.

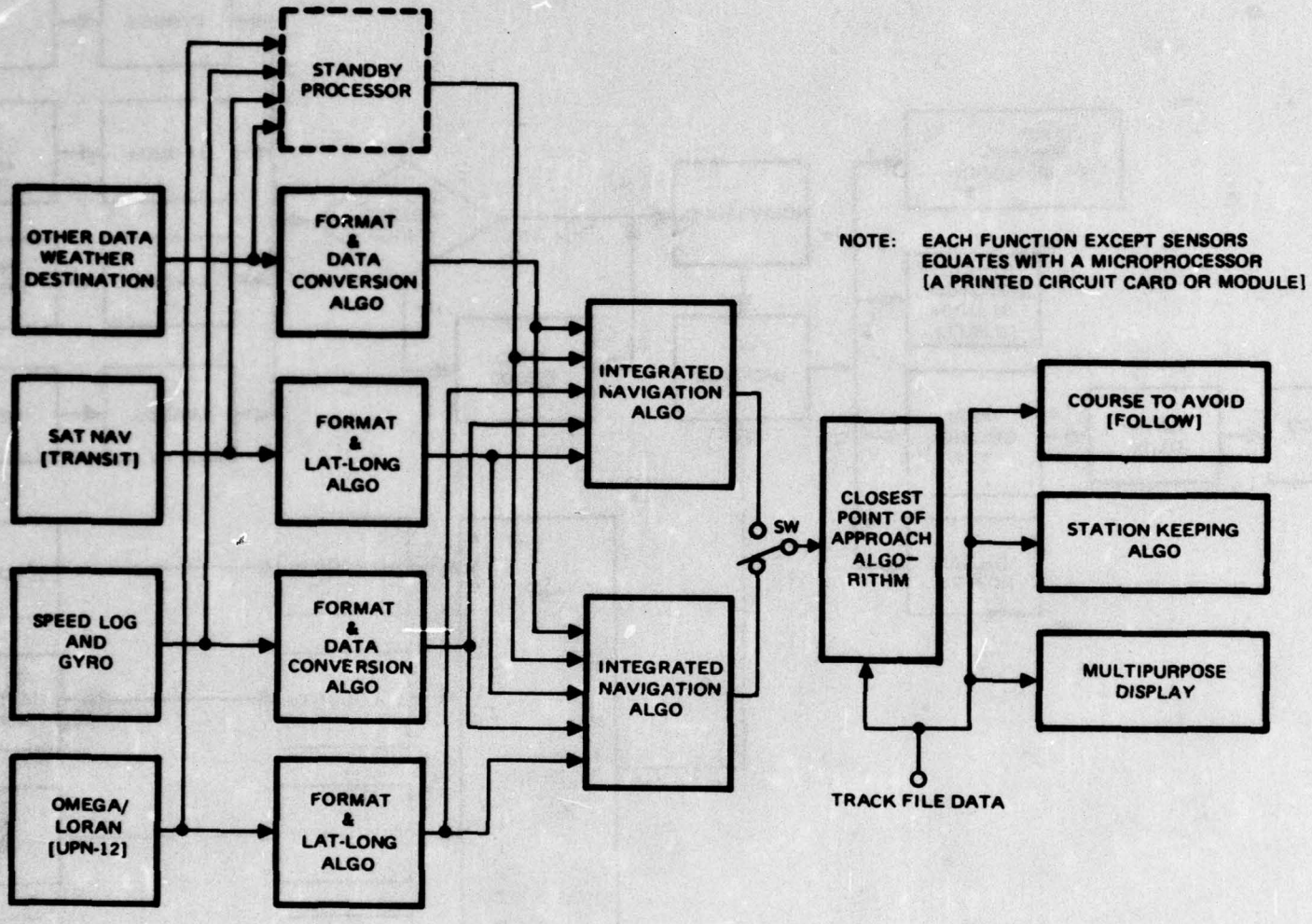


Figure 14. Navigation system.

ARCHITECTURE TRADEOFFS AND DESIGN

The selection of an optimum system architecture, whether based on a multiprocessor, a dedicated distributed approach, or something in between, is a complicated and nonquantitative process. Since the potential exists to design with thousands of microcomputers, a procedure for architectural tradeoffs is required. The selection of design and architecture must be based on the concept of computing resources. This concept involves the process of quantifying the resources that are required to implement an algorithm. Among the critical parameters to be quantified are the following.

Time. Each algorithm to be implemented requires a certain processing time, which can be measured in terms of the number (N) of long instructions or operations plus the number of short instructions or operations in the relationship

$$T = NT_s + NT_\ell,$$

where T_s = time for execution of short instructions

T_ℓ = time for execution of long instructions.

Storage resources. A measure of storage resources, S, is important for estimating cost and performance. In terms of its components,

$$S = S_p + S_k + S_{ws} + S_o,$$

where S_p = program memory

S_k = memory for constants

S_{ws} = working storage

S_o = overhead.

Input-output resources. The handling of input-output functions requires either hardware or software resources which can seriously impact the partitioning of a system or sequence of algorithms. The parameters that are to be estimated include the number of input-output operations or channels required and the maximum response time permitted for processing an input-output channel.

Each of the above parameters (time, storage, and input-output resources) is applied to the algorithms in documented form, as follows.

Mathematical form: The equations that are to be implemented.

Computation flow chart: The flow chart that describes the sequence or concurrency of the processing.

Simulation: The verification of the algorithm in the form of a program such as FORTRAN, APL, or SIMSCRIPT.

Other parameters which are potentially useful are subroutines, stacks, data bus protocol and timing, and other resource related design parameters. A potential algorithm specification is documented in appendix C. Once a specification is invoked, the procedure of design can be effectively automated and verified.

DESIGN AUTOMATION

A fully integrated design approach is required to effectively design multi-microcomputer systems. The procedure involves

Functional simulation (FORTRAN, APL, etc)

Queuing (SIMSCRIPT, GPSS, etc)

Reliability (GEM, other reliability programs)

Availability

Cost (economic evaluation programs)

In the past, the feasibility of an integrated design process, from algorithms to implementation, was a difficult manual process. But with a quantification of algorithms in units common to hardware and software capacities, an integrated, fully automated (with man in the loop) design procedure, as shown in figure 15, is possible.

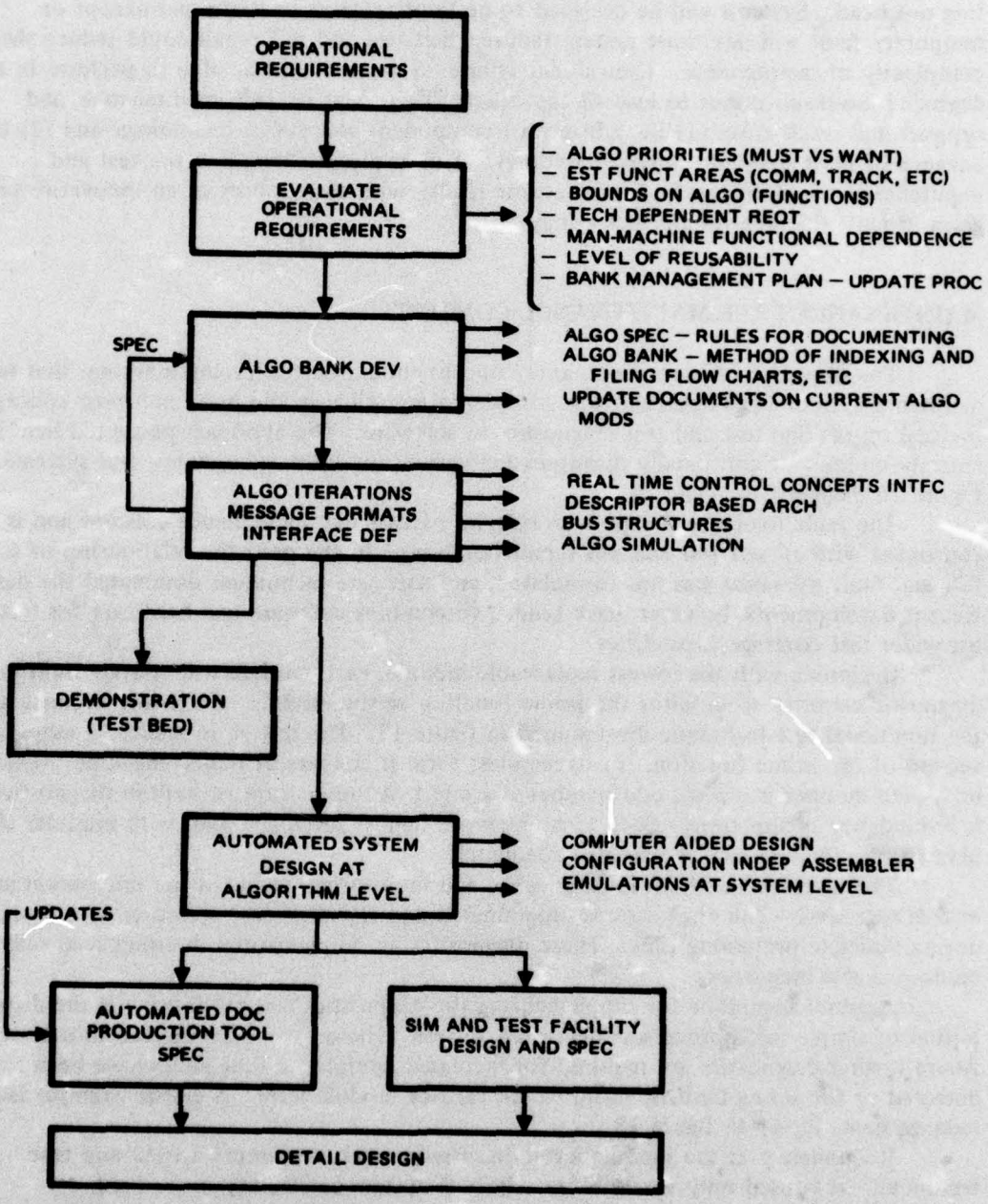


Figure 15. Design automation flow chart.

FAULT TOLERANCE, SELF TEST, AND SELF REPAIR

Design objectives based on new concepts of partitioning and using one-chip microcomputers will stress simplified maintenance, increased system availability, and lower operating overhead. Systems will be designed to be fault tolerant so that a permanent or temporary fault will not cause system failure. Self test and self repair could reduce the complexity of maintenance. Even under failure, systems should be able to perform in a degraded mode, so as not to lose all capability. This focus on test, maintenance, and support will result from (1) the advance of component and device technology and (2) the advance of fault tolerant system technology. This section will explore the test and maintenance possibilities which can become reality with the support of an innovative program in fault tolerant system technology.

MAINTENANCE-FREE MAINTENANCE CONCEPT

The objective of the maintenance concept should be to develop a system that is maintenance-free. This approach was considered infeasible in the past, and most concepts focused on off-line test and test diagnostics in software. The approach presented here is entirely on-line: it continually diagnoses the various modules, subsystems, and systems. Figure 16 diagrams the concept.

The fault tolerance design is an integral part of the maintenance concept and is consonant with all self test and self repair functions. In the past, the relationship of self test and fault tolerance was not formulated, and software techniques dominated the design. Recent developments, however, have tended to combine software and hardware for test, for wider test coverage capabilities.

Beginning with the lowest replaceable module, each module will contain built-in diagnostic circuitry to monitor the prime function of the module. Such test circuitry uses the functional test technique diagrammed in figure 17. The test is, in reality, a simplified version of the prime function. In its simplest form it consists of parity checking. (Count bits; even number is a zero, odd number is a one.) Another form of built-in diagnostics is redundancy in functions. Such techniques are highly developed and, with modules clearly identifiable, the test can be designed adequately.

The next level of built-in diagnostics and monitoring occurs at the microprogram or software level. The diagnostic is programmed and repeated each iteration of a function during available processing time. These diagnostics are implemented in either read only or random access memories.

Another technique for supplementing the diagnostics and monitoring is the distribution of simple microprocessors dedicated to test. These can be introduced into systems where further diagnostics are required for increased confidence that faults have been detected or for aiding fault isolation to the card or module level. A classic example is a radar system, shown in figure 18.

Redundancy at the module level, discussed earlier, represents a tried and true technique. It is used only where extremely high system availability is required.

Planned degradation by distributed function partition provides the technique whereby system capability is merely reduced somewhat while modules are being replaced.

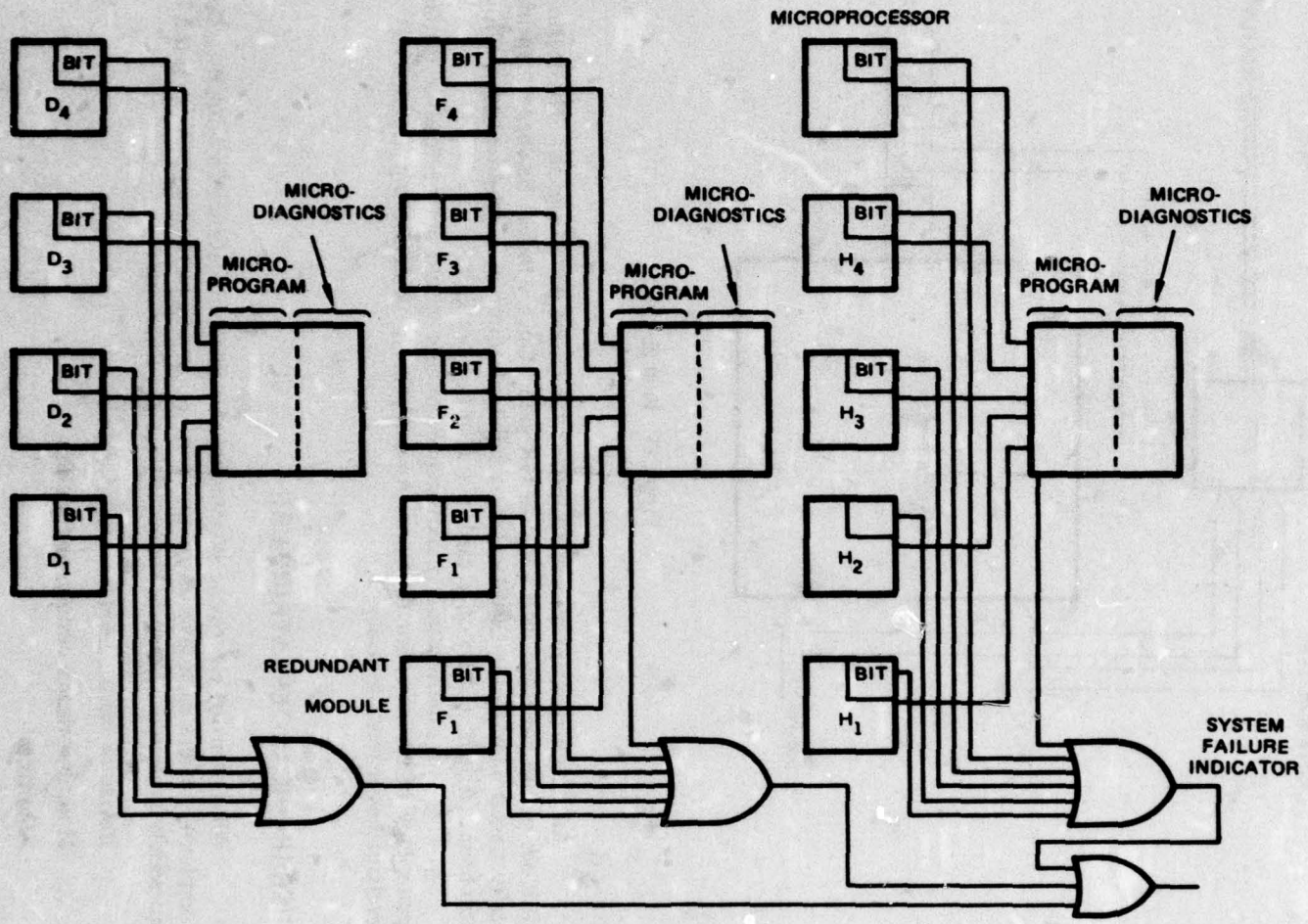


Figure 16. System maintenance concept.

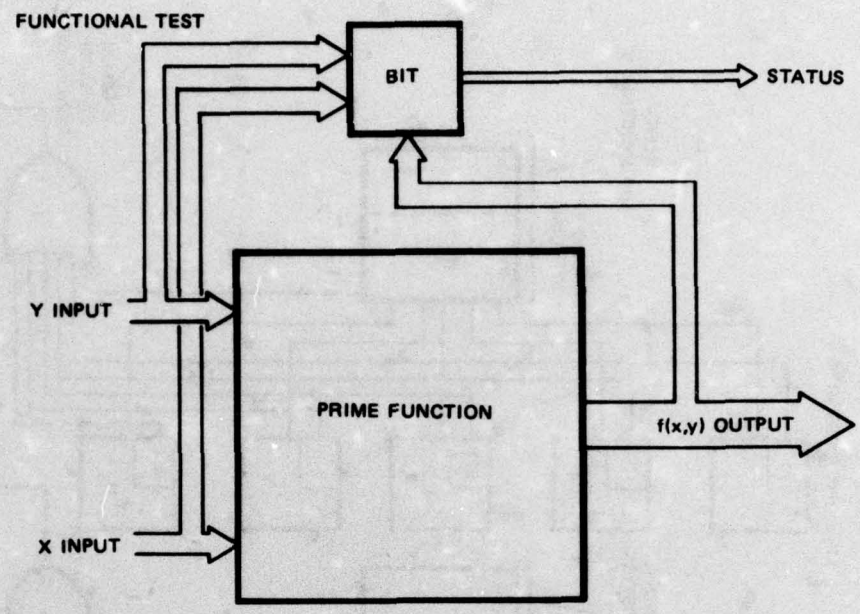


Figure 17. Built-in test.

All diagnostics are integrated into status (front panel) signals. The status bits of all modules are collected into a logical OR gate. Once a failure has been identified, it is easily traced back to the card level by a series of counters which identify specific card numbers. (A light (LED) on a card may also be feasible.)

This "no maintenance" maintenance concept would not be feasible through the technology of the past, but the conceptual approach of identifying functions at various levels makes it feasible today.

HIERARCHICAL TEST STRUCTURE

All designs of electronic systems involve different levels of modularity, test, and complexity, where each level is unique in function and technology. Among the levels that are widely accepted (shown in figure 19) are the following:

Device or circuit level

Module, printed circuit, or function

Algorithm

Subsystem

At each level, a particular architecture can be identified which represents the connections needed to perform the composite function at the next level. This hierarchical structure applies to the fault tolerant, built-in test design as well.

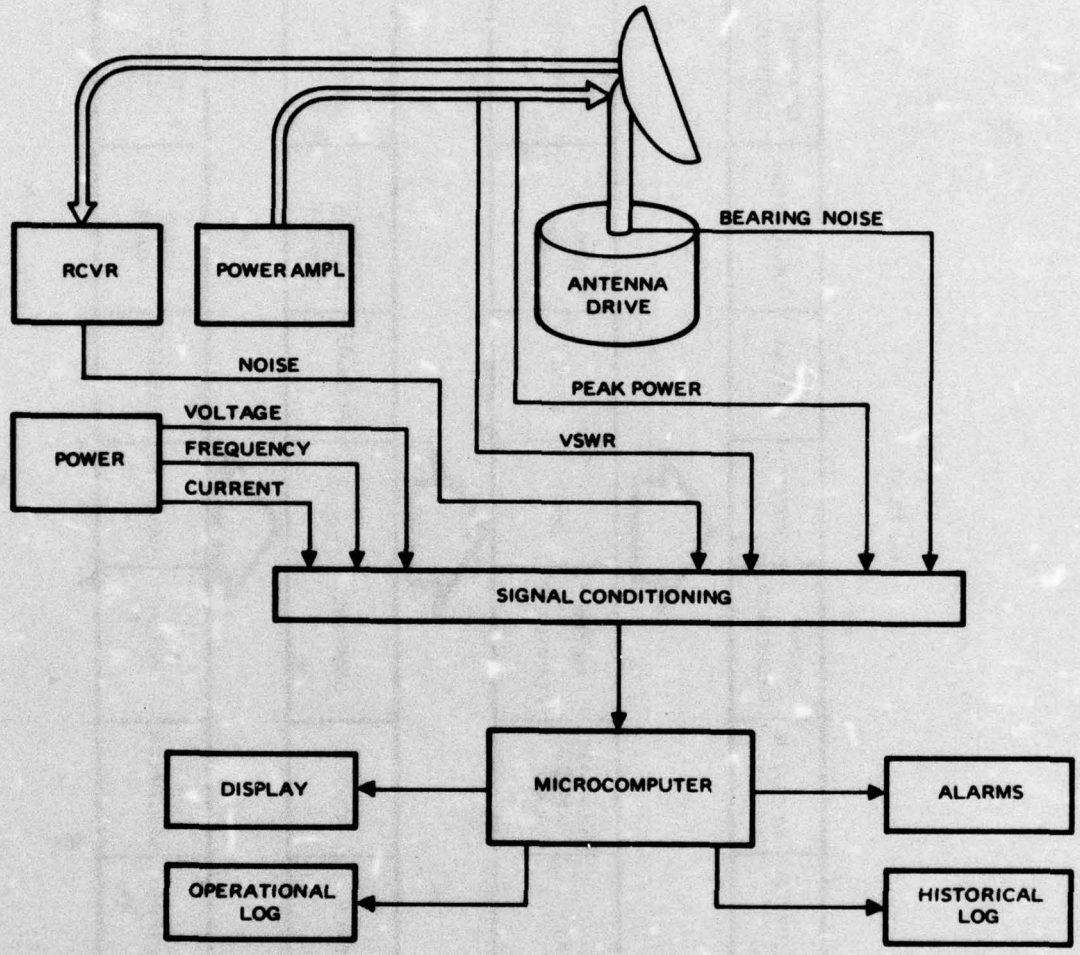


Figure 18. Microcomputers in radar system performance monitoring.

The problem of fault tolerant maintenance design is different at each level. The device or circuit level involves gates, resistors, voltages, etc. At the function level, represented as composites of devices, the concern is with checking the composite function. Algorithms, the next plateau of the hierarchy, present certain unique self test and self checking problems. They may be addressed in software as well as hardware. Each level of test can draw upon the next level below. At the subsystem level, the recovery procedure will be driven by software (control), but will be dependent on other built-in capabilities to test, diagnose, and repair.

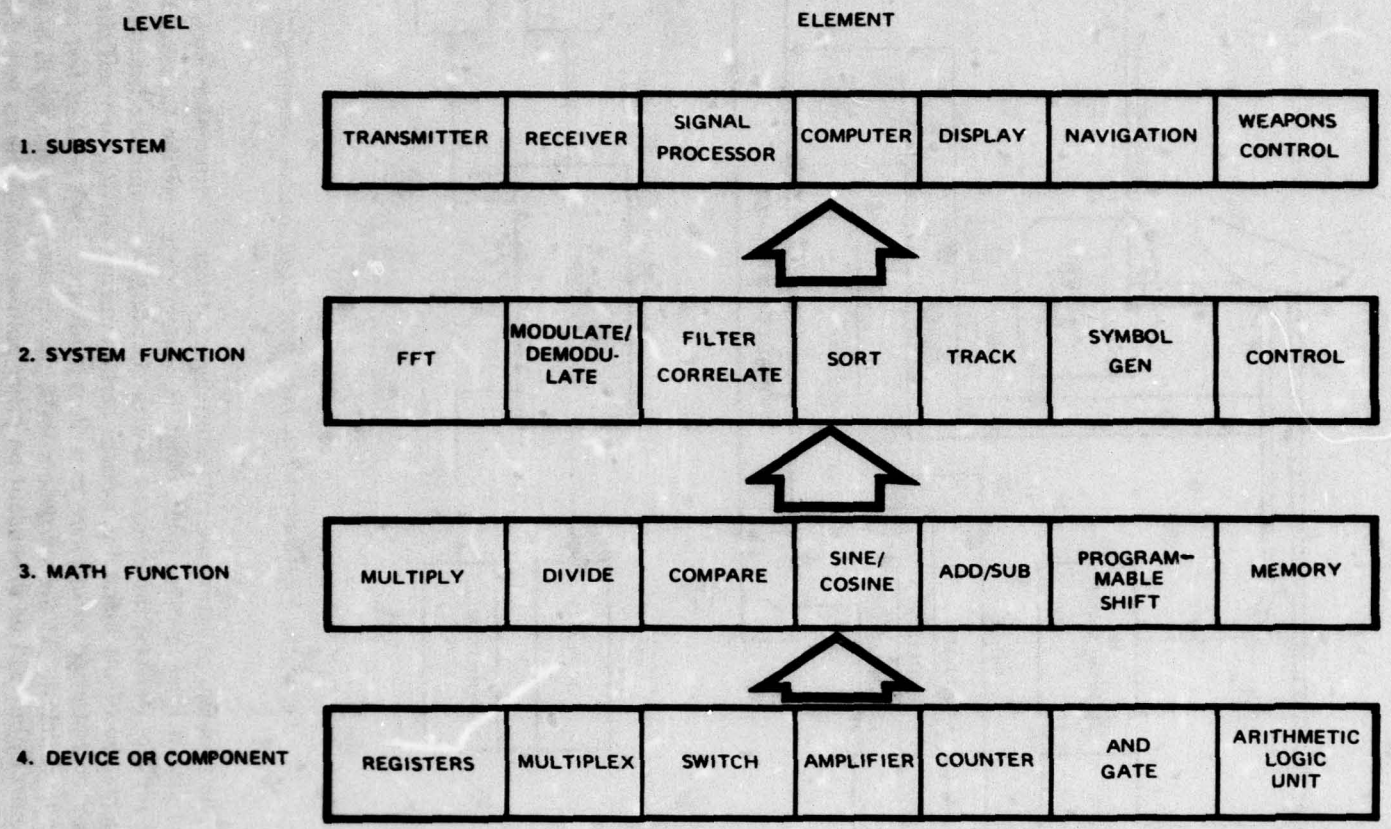


Figure 19. Levels of modularity, test, and complexity.

N-DIMENSIONAL ARCHITECTURES

The hierarchical structure can be partitioned into three or more architectural forms. Among the most commonly understood are (1) a data or functional processing architecture, (2) a control architecture, and (3) a test and maintenance architecture (figure 20). Each of these architectures can be selected independently in meeting the system specifications. For example, the data bus structure is selected to meet a throughput requirement. A control bus structure can be selected to meet a requirement of flexibility and reconfigurability, and the testing and monitoring structure is selected to meet an availability requirement. Though there is interaction between the structures, the recognition of independent architectures permits a reduction of complexity and an increase in effectiveness. These structural capabilities have only recently been realizable through the use of new circuit designs. Specific capabilities include the following:

Bus input and output circuits are designed to prevent loading down a bus if failure occurs. In the past, a failure could load down a bus and prevent simple diagnostics to the failed unit.

Built-in test circuits can be isolated from the functions they test. This has been a critical block to pursuing built-in test.

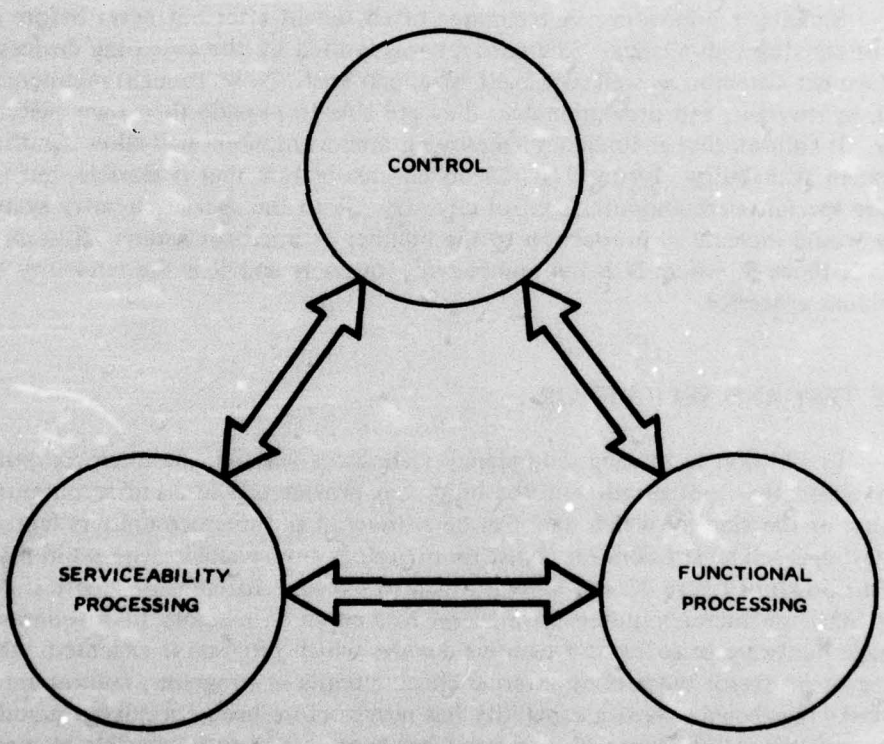


Figure 20. N-dimensional architecture.

Microcomputers are programmable test circuits and can be ubiquitously applied. Furthermore, software test programs can be implemented in firmware and isolated from the functional level.

REDUNDANT MICROCOMPUTERS

The use of redundancy at the microcomputer, algorithm, or functional level is an interesting concept with potentially high payoff. Though redundancy is not new, it has met with only limited success at the circuit or device level. The device of the future, however, is a microcomputer, and the use of redundant microcomputers offers some interesting concepts such as the following examples.

Quad redundant microcomputer design. In redundancy design, a major question centers on the voting and switching circuitry, which must be at least an order of magnitude more reliable than the units that are used redundantly. A unit which is programmable, such as a microcomputer, can be designed in configurations that determine what has failed and how to maintain the necessary functions. Such quad redundant configurations will require significant basic and applied research to determine the control, busing, and I-O protocol for new designs. Figure 21A shows a quad redundancy design which can and will be applied.

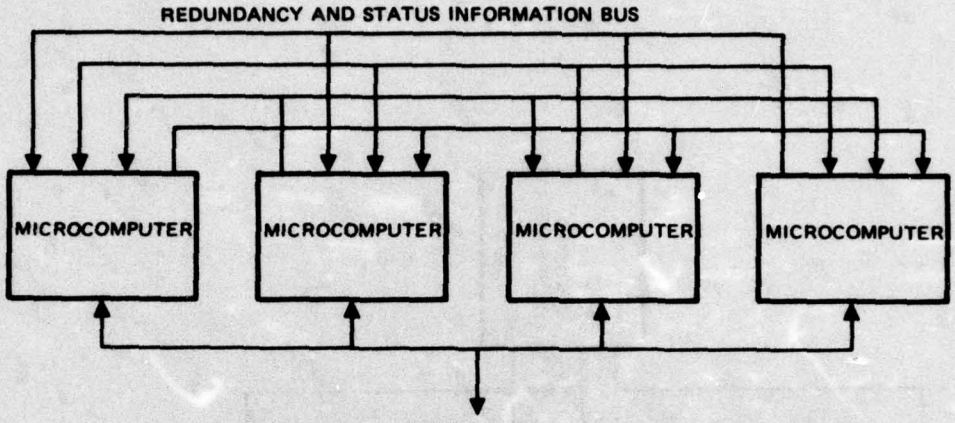
Standby redundancy. A technique often sought after but never before practical is that of standby redundancy. The concept was limited by the switching devices and the replacement schemes, as well as weight, size, and cost. Now, though, microcomputers are small, lightweight, and programmable; they are able to provide their own switching capability. It follows that redundancy employing microcomputers will allow significant increases in system availability. Figure 21B reflects an architecture that is feasible, but which would require special microcomputer control circuitry. With the special circuitry systems, reliability would increase in proportion to the number of microcomputers. System reliability, R_S, is N times R, where N is the number of processors and R is the reliability of an individual processor.

SELF TEST AND SELF REPAIR

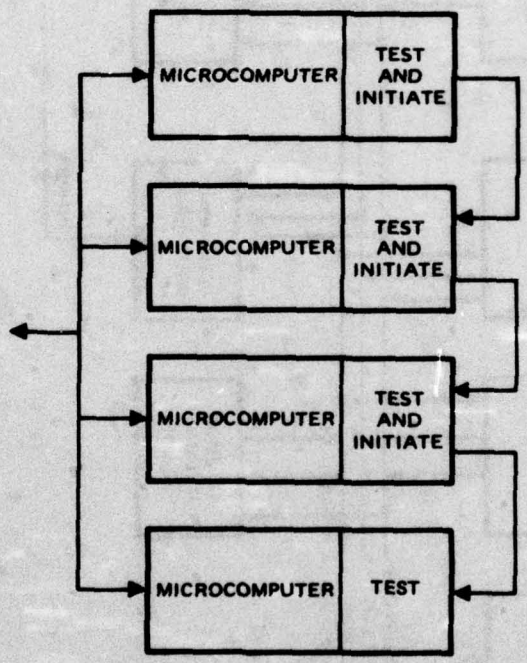
In addition to making redundancy techniques feasible, the microcomputer allows many alternative configurations to be built that provide self test and reconfigurability. Because of the ease by which data can be redirected and microcomputers reprogrammed, self test and self repair concepts must be rigorously investigated. The multi-microcomputer system shown in figure 22 can be controiled to execute different or identical programs easily since all microcomputers have access to a common memory on a round-robin basis. A single hardware or software executive decides which program is executed. By simple testing of processor outputs by internal check circuits or programs, failures are detected and tasks reassigned. Such a capability has never before been considered practical on any basis. Yet the simple approach presented has been shown to be feasible by many researchers.[5]

---

[5] Rand Corporation Report R-1011-PR, Air Force Command and Control Information Processing in the 1980's: Trends in Hardware Technology, by R Turn, October 1972



A. QUAD MICROCOMPUTER REDUNDANCY



B. STANDBY REDUNDANCY

Figure 21. Redundant microcomputers.

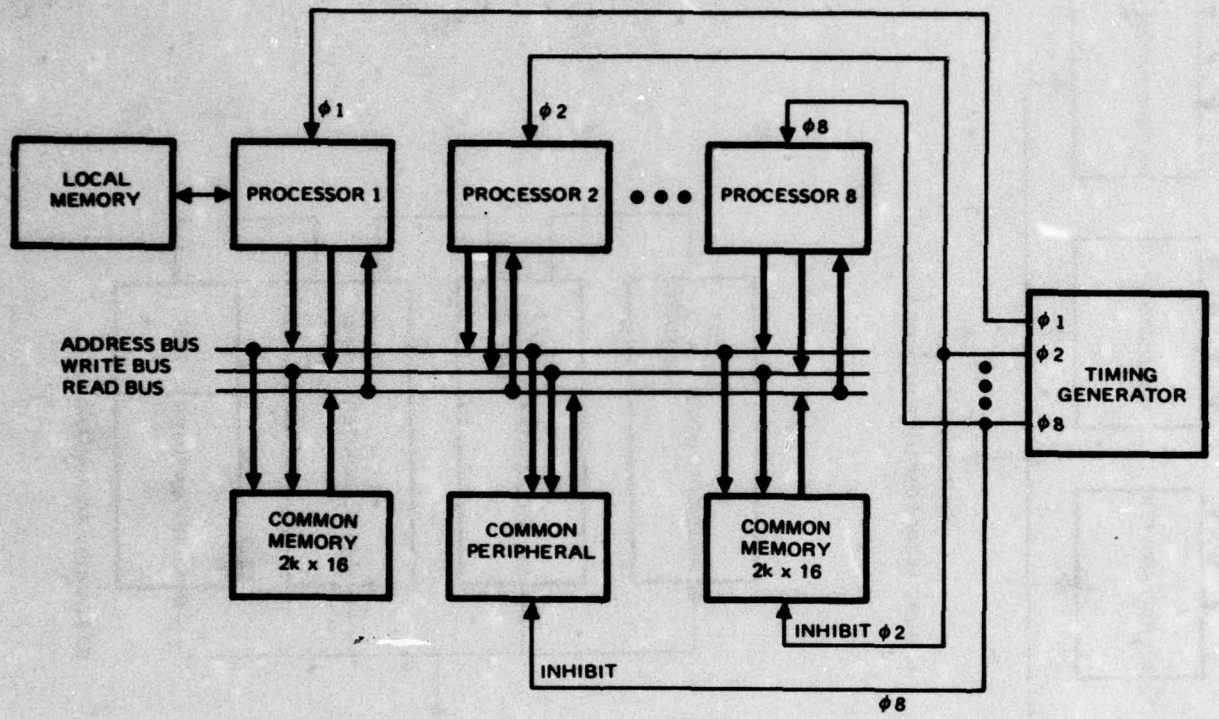


Figure 22. Distributed microprocessor.

APPLYING MICROCOMPUTERS

The impact of the advance of microelectronics technology on Navy system design cannot be underestimated. Technological research must be directed to allow the timely and effective application of this new technology in Navy systems. This section reviews the basic findings and presents some guidelines for future action by the Navy.

The microcomputer of the future will be a single chip (silicon component) with the capability of present-day minicomputers. In general, it will be a 16-bit processor with 4k words of memory and possibly with select hard-wired logic functions such as multiply. Its design will use new external control methods such as program counter control. The computer on a chip will be commonly available soon. Families of complete, standard computer chips will meet broad classes of applications. Special purpose functions such as multipliers and smart memories will be commonplace. New concepts of design, however, will not always use the microcomputer in the most efficient manner. The most effective use of microcomputers will be in providing parallelism in systems. For high maintainability and availability, new approaches to system design (eg, fault tolerance, self test, and self repair) will be feasible.

A family of application dedicated processors will be feasible. Thus high performance is obtainable for large computational needs. Examples include interactive terminals, track correlators, and linear programming processors.

The design of systems will revert to that characteristic of analog systems, with each component performing a given function in sequence. The effect will be a simplification of system control.

System design architecture can be partitioned into several unique and independent structures. For example, there will be separate functional architecture, a control architecture, and a maintenance and test architecture. Additional architectures are feasible and can be superimposed.

Interfaces will be standardized, with IEEE 488 and MIL-STD-1553 evolving as the principal bus standards.

Software will be relegated to the design of control programs and high order languages. Functional software will be dedicated to hardware by the semiconductor manufacturer. Direct higher order machines will be practical.

Design of systems will address fault tolerance criteria since the performance goals will be met by a variety of hardware configurations.

AREAS OF IMPACT

The use of fully distributed processors which are made up of a million or more microcomputers will bring us closer to the creation of artificial intelligence, ie, the thinking machine. The promises of thinking machines, theorem proving, learning systems, and many other sophistications in the fields of computing have never reached fruition, primarily because the computational requirements are extremely high. With the development of the microcomputer on a single chip, the computational potential exists to implement artificial intelligence techniques. Current efforts, though elementary, will be expanded to meet various military and commercial needs. The advance will result in capabilities never before thought possible.

Another logical application of microcomputers would be in the field of robotics. The robots of today are simple machines performing simple tasks, but this capability will

be expanded to achieve new, complex robots which can perform more intricate, more sensor dependent tasks. This achievement will result from the use of the single chip microcomputer distributed in robotic applications.

An extension of microcomputers to general purpose processors is not different from all that is implied in the previous section. This will result in the culmination of design concepts into new distributed computing systems. The fundamental calculator will be backed by a high performance computing system able to solve major problems.

The key to success in the use of a million-microcomputer device is that it provides the ability for developing fail-soft, self test, and self repair systems. New concepts in maintainability and in new component and device technology will result in increased reliability and maintainability of systems.

Unique uses of these new microcomputer concepts will continue to increase. Those concerned with Navy systems engineering should be highly informed about the design and possible application of present and future microcomputers.

As the microcomputer becomes more and more complicated through its natural evolution, the systems engineer will be required to cope with its increasing complexity. It will be necessary for the engineer to know and understand the advancements made in microcomputer technology if he is to take advantage of their many uses in system engineering.

The multitude of possible uses for microcomputers is overwhelming. It may first appear that specific applications in a system would be simple and easy to implement, but applications would in fact be difficult if not impossible without a qualified systems engineering staff to design and evaluate them and their system functions.

It cannot be overemphasized that the Navy must use the best systems engineering concepts available to take advantage of the advances made in microcomputers. The systems engineer must stay abreast of future developments in the very complex and rapidly changing field of microcomputers.

RECOMMENDED RESEARCH PROGRAM

As the result of this projection of components and system technology, it is recommended that the following research program areas be pursued.

1. Fault tolerant system design methodology. The Navy's long-range objectives include simplified maintenance, increased system availability, and lower operating overhead. To achieve these objectives, systems must be designed to be fault tolerant—able to perform in a degraded mode by using self test and self repair techniques to correct malfunctions. This capability is a critical requirement in all command control systems ashore and in the fleet, and can yield significant savings in support costs and increases in system availability.

This program will be directed at the formulation of a basic fault tolerant methodology. It will include the development of a common lexicon, ie, definitions and terminology. The program will produce a self test and self repair methodology which includes a hierarchy of test and repair architectures. The program identifies test structures at the device, component, algorithm, and system levels. It must identify tradeoff analysis tools. This program will address the role of software in self testing, the methodology of developing built-in test software, and the use of microdiagnostics which are implemented in firmware. Critical to this program is the designing of fault tolerant systems by superimposing the fault tolerant structure on the existing system architecture.

The current efforts in fault tolerance address only selected problems, such as Draper's work on fault tolerant buses, Honeywell's effort on Petrie nets, and Stanford's efforts on the self diagnostic computer. This program, in contrast, must integrate the area of fault tolerance into a unified design methodology. Only through the development of such a unified methodology will the Navy be able to afford fault tolerance in its systems.

2. Distributed file management systems. Navy systems involve the processing, storage and interaction of large data files. In particular, the systems demand file data on tracks, ships, logistics (spares, etc), training, calibration, specification, activities, and historical data. In addition, data files must be distributed to the levels where they are used and must be centralized for high level decisionmaking. Though distributed file management techniques appear to be the answer to the Navy's problems, this technology is in its infancy, and the basic foundations of distributed file management systems are not understood.

This technology requires advances in file structures, dedicated (back-end) processors, distributed architectures and networks, interaction file up-date methods, communication structures and architectures, and other new technological advances. File structures involve the classification of data by characteristics usage (update rates, subsets, etc), storage and transmission requirements, and file search processors (associative processors). Distributed file management systems involve networks such as the STAR, RING, and REDUNDANT (MATCHED) network, but other networks must be identified and their characteristics evaluated. Current architectures appear inadequate, but with new advances in mass memories and dedicated file management processors, new architectures are sure to be recognized. New architectures, methods to trade off architecture, and file management techniques that use back-end processors are principal developments of this research.

3. Distributed microcomputer system design methodology. The rapid advance of microcomputer technology has resulted in small but computationally powerful computers that can be used in innovative design approaches leading to simple, more effective systems with lower life cycle cost. The design of weapon systems, support systems, and other shore equipment requires the development of a foundation of distributed microcomputer system design. This program shall include design of dedicated processors and the design of brainlike computer structures. Current design approaches using methods based on conventional computers are not adequate for use with microcomputers; due to their small size, new microcomputer system architectures must be pursued.

The program will classify the nature of microcomputer design methods. For example, extending classical computer technology to the design of systems involving only a few processors is well understood. The use of 1000 to 10 000 microcomputers for dedicated processing developments such as a file search processor, linear programming computer, or regression analysis computation is a major multi-microcomputer design category. Another major area is the development of brainlike processors for pattern recognition, artificial intelligence, and robotlike controllers. These areas of microcomputer systems design are fundamental to Navy problems in maintenance and support, and efforts must be accelerated to effectively utilize the capabilities of large scale integrated (LSI) microcomputer technology.

4. Structured design methodology. The design of future Navy systems must be accomplished by integrating all design parameters and trading them against performance and support objectives. This approach to design requires the formulization of a structured design methodology, the interactive use of computer aid design and decision tools, and integration of all facets of design. A structured design methodology is the key to improved reliability, simplified maintenance, and lowered life cycle cost (both lowered acquisition cost

and cost of ownership). This program will address system design at the algorithm level–a key to successful design with microcomputers and to the other advances in component and device technology.

Fundamental work by Draper Labs and others on high order software has shown the feasibility of a set of high order design rules and axioms. This approach can be structured for the design of systems involving both hardware and software and can result in the mechanization of the methodology of design. Developing the axioms and integrating design models such as functional simulation, queuing model, life cycle cost models, and reliability and maintainability (R&M) models will result in a powerful and cost effective design methodology that will reduce the cost of ownership and increase the performance of Navy systems.

An important aspect of this program is the development of algorithmic architectural tradeoffs. Recent efforts by NELC indicate that algorithms can be partitioned to achieve functional and mechanical package equivalence. This program shall pursue the relationship of algorithms and their implementation by the development of a resource model. This can result in a new methodology of design for Navy systems.

The Navy must employ interactive man-machine systems to meet our supportability objectives. The complexity of current test problems requires the optimum man-machine mix if test and diagnostic effectiveness is to be achieved.

A critical problem area involves the generation of test application programs, in which man and computer interact to develop test programs far more effectively than either alone. Programs should address man-machine interaction in system design. They should identify those functions to be performed by the man and the machine to achieve synergism in the design procedure. In addition, there is a need to explore the use of voice control and response, touch or manual, and visual communication interfaces. The research shall address interactive training and operational systems.

5. Application languages. The future for the application of microcomputers will be broad. Many applications, unique and currently not considered practical, will be discovered. This diversity of application will require application oriented software languages. For example, a language for control-processor, autopilots, etc, would be desired. A language for test and diagnostics would be extremely desirable for microcomputers used in built-in test, monitoring, and self repair configurations. A program is required that addresses the categories of applications, the differences and similarities, and other basic foundations of application software. This program will also address software standardization concepts of the future microcomputer era that are consistent with the development of DoD standard high order language efforts.

6. Modular programming. The requirements of Navy systems are such that classes of application problems exist and that standard program modules can be selected to address these applications. The Navy must continue to pursue modular programming and a set of system primitives. In particular, algorithms which form the basis of algorithms at the system, subsystem, and printed circuit card and component level shall be identified and modularized. The methodology of selecting programs and their characteristics will be pursued. This program must be integrated with the efforts on structured design methodology, distributed microcomputer system design methodology, and other closely related efforts.

7. Validation and verification. Software and system certification, which involves knowing that a system meets the requirements for which it was designed, is a major Navy concern. It is an area that will require greater consideration as a result of the introduction of microcomputer technology and its added complexities. A program which addresses the

role of test verification from requirements through design to final acceptance test is needed. A technological foundation which is the basis of system validation and verification must be developed.

8. Design techniques. The use of multi-microcomputers will require complicated system engineering design techniques. The Navy can take advantage of the vast improvements that are being made in microcomputer technology only by staying abreast of those advances. Navy system engineers will thus be better able to cope with the increasing complexity and to maximize the use of microcomputers. The Navy should initiate separate studies to determine the problems, capabilities, and future guidelines for multicomputer systems design. These studies should be directed toward the areas of sensors, weapons control, flight control, command control, and communications.

CONCLUSIONS

The computer on a chip will be commonly available soon.

Complete, standard families of computer chips will meet broad classes of applications.

Special purpose functions, multipliers, and smart memories will be commonplace.

New concepts of design will evolve in which microcomputers are used inefficiently but effectively in achieving parallelism in systems.

New approaches to system design for improved maintenance and availability (eg, incorporating fault tolerant, self test, and self repair features) will be feasible.

RECOMMENDATIONS

1. Initiate separate studies to determine the problems, capabilities, and future guidelines to design multi-microcomputer avionics.
2. Conduct such a study in each of the following areas:

   Sensors

   Weapons control

   Flight control

   Command control

   Communications
3. Explore new concepts of system design in which microcomputers are incorporated. Emphasis should be on fault tolerance, self test and repair, microcomputer primitives, and dedicated system design.
4. Undertake follow-on studies to project and quantify capabilities that may be achieved through the new systems design concepts in flight control, weapons control, and sensor integration.
5. Explore the impact of new component microcomputers on robotics and in artificial intelligence applications. The potential applications of this new technology could be astounding.

REFERENCES

1. Moore, Gordon E, Progress in Digital Integrated Electronics, International Electron Devices meeting, Washington DC, December 1975.

2. Honeywell, Inc Report AFAL-TR-73-226, All Semiconductor Distributed Aerospace Processor Memory Study, by MD Johnson et al, August 1973.

3. Rand Corporation Report R-1012-PR, Air Force Command Information Processing in the 1980's: Trends in Software Technology, June 1974.

4. University of Illinois Report UIUCDCS-R-74-638, Program Speed Through Concurrent Record Processing, prepared for the National Science Foundation by RE Strebendt, October 1974.

5. Rand Corporation Report R-1011-PR, Air Force Command and Control Information Processing in the 1980's: Trends in Hardware Technology, by R Turn, October 1972.

APPENDIX A

MICROS, MINIS AND NETWORKS

This paper was prepared by Edward J McCluskey, of the Digital Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, California. It was presented at a meeting on "Twenty Years of Computer Science," Pisa, Italy, 16-19 June 1975. The work was supported in part by National Science Foundation grant GJ-40286.

ABSTRACT

This paper presents summaries of the characteristics of four multiprocessor systems: the Plessey System 250, the Berkeley Prime System, the Carnegie-Mellon C.mmp, and the BBN Pluribus System. Detailed bibliographies on each system are included.

INTRODUCTION

My interest is in systems composed of interconnected microprocessors. While these are, in a sense, networks, they are characterized as being tightly, rather than loosely, coupled. There are two general classes of tightly-coupled microprocessor systems: distributed-intelligence systems and multiprocessor systems [Raphael, 1975]. They differ in that the task assignments in a distributed-intelligence system remain fixed while multiprocessing systems are designed specifically to handle a dynamically varying set of tasks such as is present in a time-sharing application.

Since I am not aware of any multiprocessor systems constructed of microcomputers, the approach I have taken here is to present a discussion of multiprocessors which use either minicomputers or specially-designed processors. This is done because of my belief that it is out of the experience gained with these systems that the insight to design microprocessor multiprocessor systems will come. The interest in such systems stems from the economic advantages to be obtained by the use of LSI processor technology and the desire to determine to what extent this technology can be used to build power computer systems.

In the following sections the details of four multiprocessor systems will be presented. It is interesting to study the techniques which are used in each of them to cope with the problems of operating tightly-coupled computers. One of these systems, Prime, was designed specifically to provide time-sharing; the Pluribus was designed for use with the ARPA network; and the remaining systems were to be very general purpose. All but the Plessey System 250 make use of standard minicomputers. Work on the Prime system has been stopped; C.mmp is operational at Carnegie-Mellon University; and the other two systems either are presently being used for production computing or will be very shortly. The Pluribus system differs from the others in that the design is very heavily oriented towards its application as a communication processor for the ARPA network. In spite of this, its designers are optimistic about its more general applicability and the possibility of replacing its minicomputers by microprocessors. It is interesting that two of the systems—Pluribus and Plessey—do not rely on interrupts for handling I/O.

Raphael, HA, Joining Micros into Intelligent Networks, Electronic Design, vol 5, March 1, 1975, p 52-57

THE PLESSY SYSTEM 250

This is a system designed to operate with from two to eight processors. Each processor has one 60-bit parallel bus which communicates directly with each main store module or peripheral module. See figure A-1.

There can be a maximum of 40 such modules connected to each bus. A store module, typically 32k of 24-bit words, connects to the processor buses via an interface unit which:

i) recognizes requests for access

ii) resolves conflicts

iii) allocates access cycles to requesting processors.

Each peripheral device has an interface unit which provides only two bus ports. Thus when there are more than two processors in the system, the peripherals must be connected through multiplexers. High activity or fast devices (backing stores) are connected to the processor buses directly (or through a multiplexer); low activity or slow devices (terminals) are connected through Serial Parallel Adapters which collect and distribute packets assembled from the serial devices [Cosserat, 1972].

The system utilizes 24-bit words for data, instructions, and addresses. There is one uniform address space: peripherals are accessed via control and data registers which the processors address in the same fashion as main store location. Thus there are no separate I/O instructions.

The processors do not have any external interrupt mechanisms and there are no separate I/O channels. Three internal conditions can cause a change of process:

i) A program trap which occurs when a process tries to execute an instruction whose effective address references a location which is not in the main store. This trap causes a "trap handler" process to initiate actions to enter the referenced block into the main store.

ii) A time-out of the processor's interval timer clock. This activates a timer process.

iii) An internally-detected fault condition. This initiates a fault interrupt sequence.

Input-output is handled differently depending on whether a slow (serial) or fast (parallel) device is involved. For serial devices the Serial Parallel Adapter acts as a buffer which stores data being transferred to or from the device. Data is transferred between the Serial Parallel Adapter and Main Store by a process that periodically polls a bit in a status register to determine which devices are ready to be serviced.

The rationale for not allowing external interrupts stems from a desire to avoid the overhead involved in process switching in a multiprocessor system. It was decided not to have functionally specialized processors in order to have a system which would be easily expandable with little software reorganization. Thus any processor must be able to execute any process. The ability to restart an interrupt process on a different processor could lead to a high overhead.

Reliability considerations require that all system modules be at least duplicated. Thus if specialized I/O processors, ie, channels, were used, at least two such modules would have to be incorporated into the system. Early designs did have provisions for such processors, but the later designs did away with the channel units in the interest of economy and in order to avoid having external interrupts.

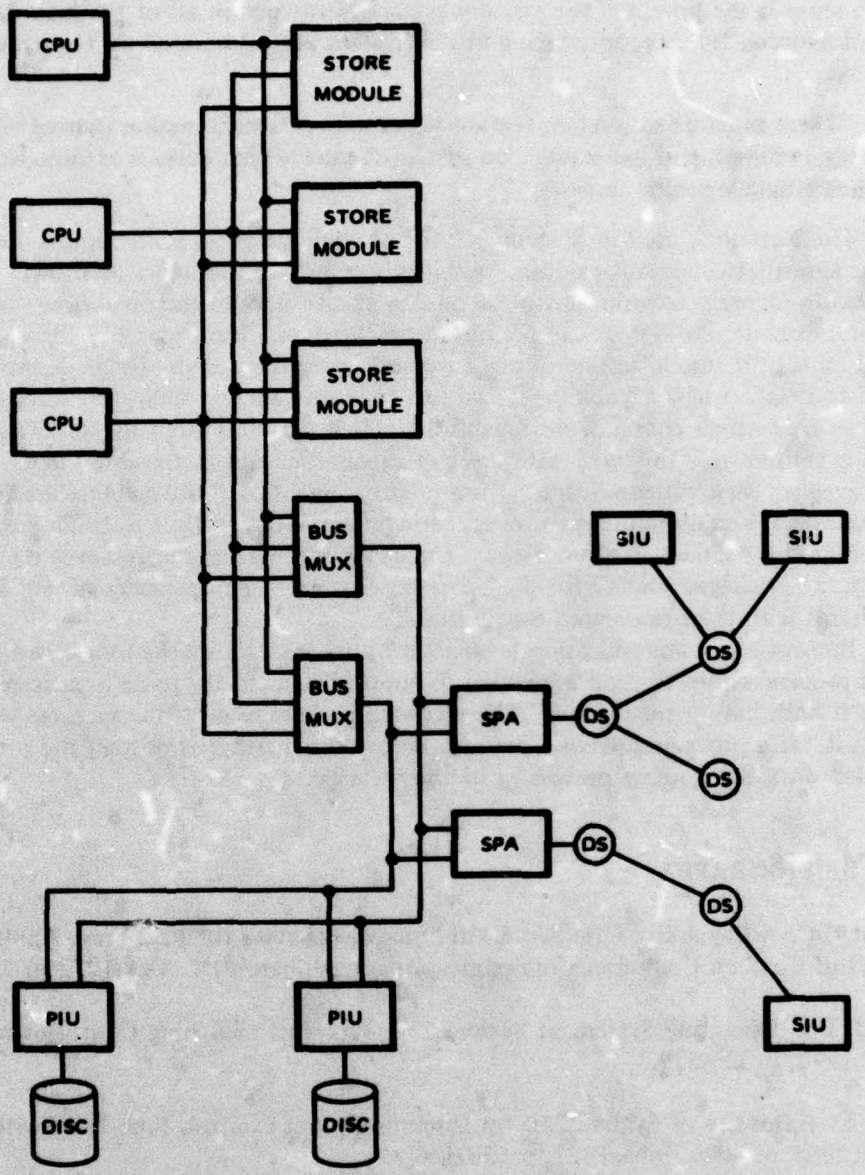


Figure A-1. Typical hardware configuration of Plessy System 250.

A basic problem in multi-processor systems in which each processor has access to all of main store is the potential for a faulty processor to corrupt all of the main memory. The goal adopted for memory sharing in the System 250 was stated by Cosserat [Cosserat, 1972] as:

"There must be no system feature which prevents information sharing where this is logically required; and conversely, no system feature which permits information sharing where this is not logically necessary."

The technique used in System 250 for providing storage protection is a universal segment identifier called a capability. A capability defines the upper and lower ends of a block wholly contained within a store module. It also defines the operations–write data, read data, execute, etc–which are permitted on the data. The capability for a process running on a CPU is held in one of eight capability registers of the CPU. A process is not allowed to create a new capability; it can just load into the capability registers from an area of storage which contains the capabilities which it is permitted to use. Thus when a process is started, it is provided with a set of capabilities and is prevented from operating with any other capabilities. Strict hardware enforcement of this discipline limits the memory locations which a malfunctioning process can contaminate and thus restricts the effects that can be caused by a fault in a processor. The capability mechanism removes the necessity of having a "privileged mode" for the operating system. The operating system is just a set of programs with their associated capabilities.

Interprocess communication is handled by means of a mechanism called a flag. When a process wishes to send a message to another process, it "posts a message on a flag for which both have a capability." The receiving process receives the message when it requests it. If a process requests a message before the message is posted, the process is suspended until the sending process posts the message.

BIBLIOGRAPHY

Cosserat, DC, A Capability Oriented Multi-Processor System for Real-Time Applications, Proc Intl Conf on Computer Communications, Washington DC, Oct 1972, p 282-289.

England, DM, Operating System of System 250, Proc Intl Switching Conf, Boston, Mass, June 1972, p 525-529.

Halton, D, Hardware of System 250 for Communication Control, Proc Intl Switching Conf, Boston, Mass, June 1972, p 530-536.

Hodges, KJH, Fault Resistance and Recovery within System 250, Proc Intl Conf on Computer Communications, Washington DC, October 1972, p 290-296.

Hodges, KJH, A Fault-Tolerant Multiprocessor Design for Real-Time Control, Computer Design, December 1973, p 75-81.

Repton, CS, Reliability Assurance for System 250–A Reliable, Real-Time Control System, Proc Intl Conf on Computer Communications, Washington DC, October 1972, p 297-305.

THE BERKELEY PRIME SYSTEM

The initial design of this system was made up of five processors, fifteen disk drives and thirteen 8k memory blocks interconnected as shown in figure A-2. The external access network, EAN, permits any of its bottom ports to be connected to any other port (top or bottom). This is the only unit in the system which is not replicated. The claim is made [Borgerson, 1972] that the EAN is designed so that any internal failure "simply manifests itself as a failure of at most two ports." Since the devices attached to the ports are all replicated and the system is designed to reconfigure out any failed units, a failure in the EAN is tolerable.

One of the processors is designated electronically as the control processor and the other processors function as problem processors and as such perform user computation. The control processor is responsible for scheduling and resource allocation. Since all processors are identical any one can act as the control processor. If the control processor detects an error (in itself or one of the problem processors), it initiates actions to cause another processor (not the one suspected of being faulty) to take over as control processor. The control processor can also be replaced upon request from one problem processor if the control processor agrees to yield or by requests from two problem processors if the control processor does not agree. After a new control processor is designated, diagnostics are run to identify the faulty unit. Once the faulty unit is discovered it is isolated and the system is reconfigured to run without it.

The operating system is realized in two separate and independent parts: the control monitor and the local monitor. The control monitor implements a set of virtual subsystems each consisting of a virtual processor, some virtual memory and a virtual disk unit which are never shared with any other subsystem. There is also a virtual communication system which allows messages to be passed between cooperating subsystems (but which requires agreement from the receiving subsystem before allowing a sender to interrogate or control it). No common memory or disk storage areas are shared between subsystems. The control monitor schedules processes, allocates memory and disk storage, and manages the virtual communication system. Two parts make up the control monitor: the central control monitor (CCM) and the extended control monitor (ECM). The CCM, which is written in a higher-level language, executes only on the control processor. It handles the centralized decisions about allocation of processors, memory and storage, as well as buffering and routing messages. The ECM, which is realized in the microcode of each processor, sets the memory address map, handles disk accesses and process swapping, and sends and receives messages.

The local monitor provides the remainder of the normal operating system functions. It controls paging including working set management and part of the address mapping. Each process has its own copy of the local monitor which runs completely within each process.

A major goal of the Prime system is to ensure that a fault does not cause one user's information unintentionally to become available to another user. The mechanism used to accomplish this is called dynamic verification [Fabry, 1973]. Every operating system decision which, if made improperly, could allow information to become available in an unauthorized way is identified and a failure-independent consistency check is performed on the result of the decision. A consistency check is said to be failure-independent of a decision if and only if there is no single fault which could cause both the decision and check to be incorrect. An example of such a decision-check pair occurs when the CCM, running on

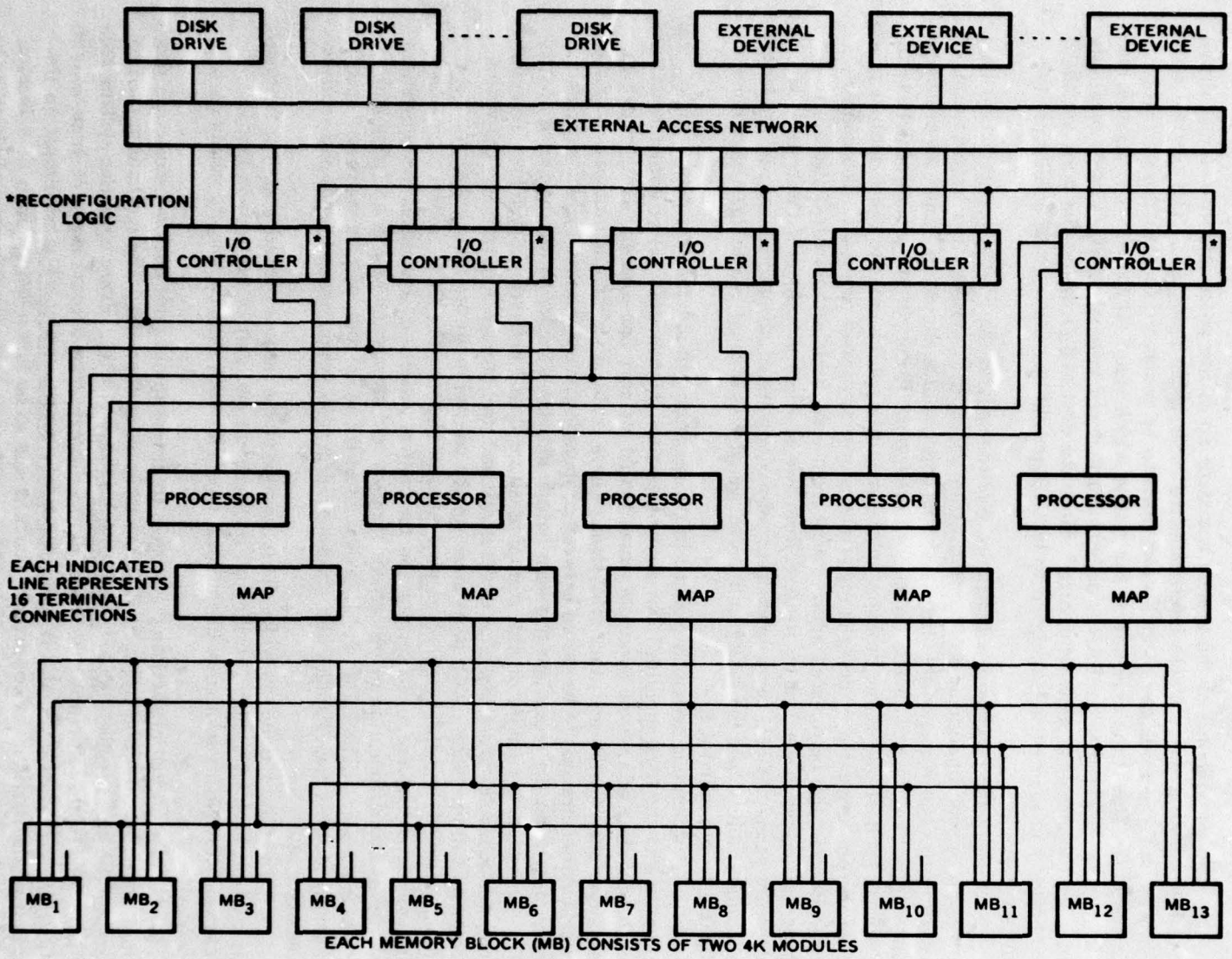


Figure A-2. A block diagram of the Prime System.

the control processor, sends a message to the ECM of a problem processor to start a process. The ECM can check whether this is a reasonable message and the check is independent of the decision since the ECM is a different block of code from the CCM and moreover is running on different hardware (a distinct processor).

More complex mechanisms are required for verification of allocation and access of memory pages and disk cylinders. For checking page accesses a lock register is provided for each page of each memory block and a key register is implemented in each processor's memory interface. All memory accesses are checked by comparing the contents of the appropriate lock and key registers. In addition each disk page has recorded with it a class-code. Each process has a unique class-code assigned to it which is recorded in the class-code register of the physical disk drive assigned to the process. The drive and page class-codes are matched whenever a process accesses a disk page.

The Prime system was designed to use commercially available microprogrammable minicomputers. Thus it has no special CPU features to facilitate multiprocessing or fault detection. Uncommon features of Prime are:

i) Each processor can only access a subset of the memory modules.

ii) No sharing of memory or storage pages is allowed.

iii) At any time all scheduling and resource allocation decisions are restricted to a single processor rather than being treated as processes which can run on any CPU.

BIBLIOGRAPHY

Baskin, HB, Borgerson, BR, Roberts, R, PRIME–A Modular Architecture for Terminal-Oriented Systems, AFIPS Conf Proc, 1972 Spring Joint Computer Conf, vol 40.

Borgerson, BR, A Fail-Softly System for Time-Sharing Use, Digest of Papers, 1972 International Symposium on Fault-Tolerant Computing, Newton, Mass, June 1972, p 89-93.

Fabry, RS, Dynamic Verification of Operating System Decisions, Communications ACM, vol 16, no 11, November 1973, p 659-668.

Quatse, J, Gaulene, P and Dodge, D, The External Access Network of a Modular Computer System, AFIPS Conference Proc, 1972 Spring Joint Computer Conf, vol 40.

THE CARNEGIE-MELLON C.mmp

This system consists of sixteen processors (modified PDP-11 processors) interconnected by a crosspoint network to sixteen primary memory modules (each having 65k of 16-bit words); see figure A-3. Any processor can access any of the memory modules with conflicts being resolved by the crosspoint network. Since the network permits simultaneous independent memory accesses several processor-memory interactions can be taking place at any instant of time. An interconnection path between a processor and a memory can be established either by a manual switch or under program control by the processor assigned the control responsibility. One possible mode of operation of this system is as a number of independent, noninteracting systems established by the crosspoint network.

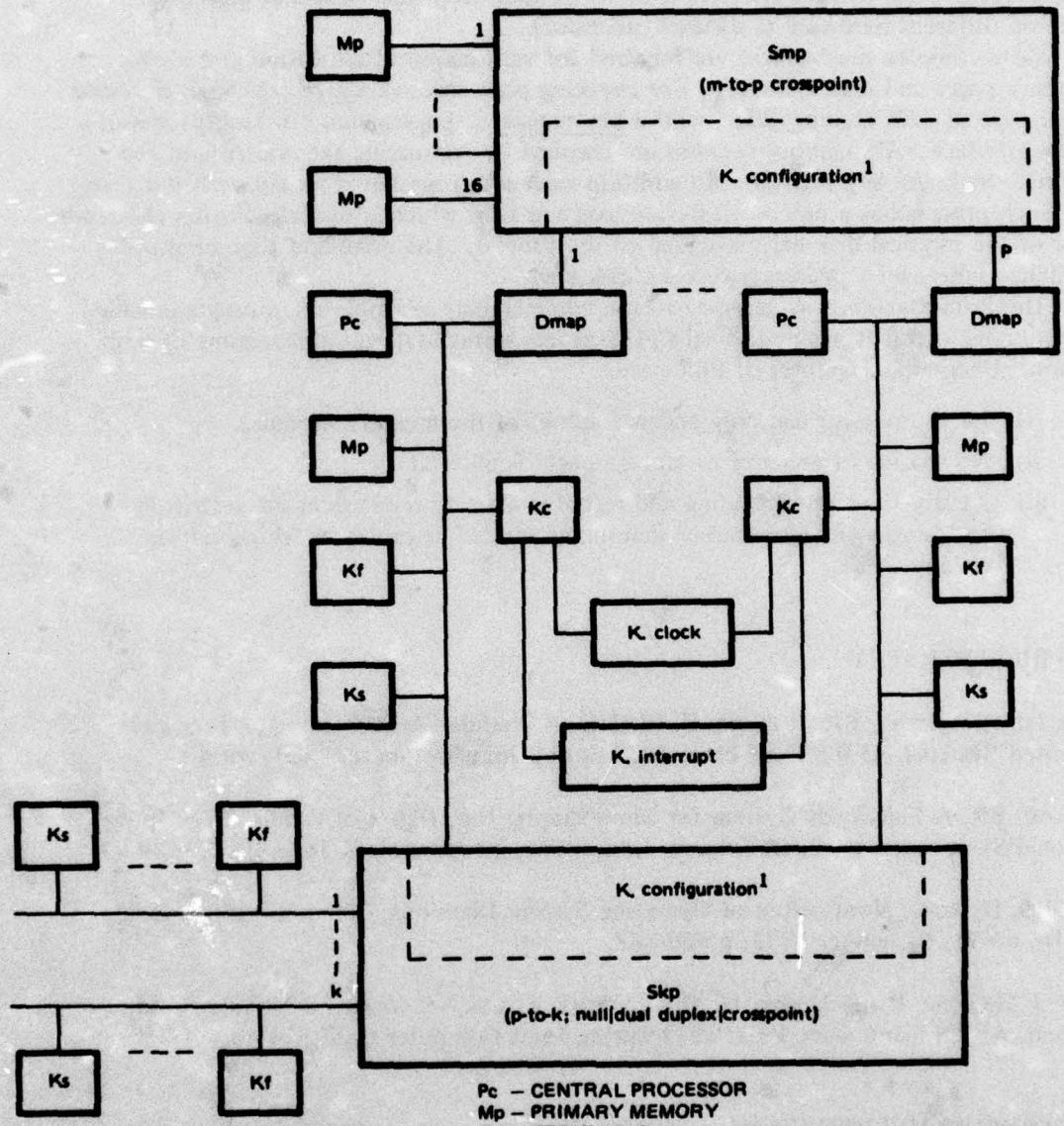


Figure A-3. Proposed CMU Multiminiprocessor computer/C.mmp.

Any processor can directly interrupt any one or group of the other processors. Each processor is actually a complete computer since it has its own local memory and local secondary storage and I/O device controllers. Also each processor has associated with it a facility, Dmap, for translating processor addresses into physical addresses for the large shared memory. Dmap makes use of thirty relocation registers which define the processor's vertical memory and which also contain some bits for keeping track of write operations and preventing unauthorized write operations.

The I/O devices are grouped onto k Unibus structures which are connected to the processors via a second crosspoint network which can also be set either manually or under program control. It is expected that this interconnection will be changed only infrequently.

Protection against unauthorized memory accesses is implemented by means of an operating system kernel called Hydra [Wulf, 1974].

BIBLIOGRAPHY

Cohen, E, Symmetric Multi-Mini-Processors: A Better Way to Go? Computer Decisions, January 1973, p 16-20.

Wulf, WA and Bell, CG, C.mmp—A Multi-Mini-Computer, Proc AFIPS, 1972 Fall Joint Computer Conf, p 765-777.

Wulf, WA et al, HYDRA: The Kernel of a Multiprocessor Operating System, Communications ACM, vol 17, no 6, June 1974, p 337-347.

THE BBN PLURIBUS SYSTEM

This system makes use of a number (typically from six to fourteen) of Lockheed SUE computers to obtain its processing power. Each processor has a 4k local memory as well as access through a bus coupler to a larger shared memory (32k–1024k words). All processors are identical and equal, and can access all of memory. The system was originally designed as a communication processor (interface message processor, IMP) for the ARPA network, but plans are presently underway to use one system as a host computer for processing seismic data. In the original design the use of the parallelism made available by having several processors was relatively straightforward since each processor is dedicated to handle one separate message.

The SUE processor has a single bus for accessing both memory and I/O. A separate module, the arbiter, controls bus access and resolves conflicts. It is possible to connect up to four SUE processors to a single bus. In this system only two processors are connected to each bus since with two buses almost no processor performance is lost while there is a substantial degradation with more than two processors on the same bus. Communication between a module on one bus and a module on another bus is handled by means of a Bus Coupler which consists of two cards and an interconnecting cable. One use of a Bus Coupler is to allow a processor to access shared memory. In this case one card of the Bus Coupler is plugged into a shared memory bus and the other card is plugged into the processor's bus. When the processor makes a memory request in the address range that the Bus Coupler recognizes, it sends a request down the cable to the memory bus. The memory

bus arbiter then is responsible for granting access to this request just as if it came from a processor directly connected to the bus. From the viewpoint of the requesting processor the memory reference appears no different from a request to a memory on the same bus except possibly for an additional delay due to the Bus Coupler. In addition to permitting accesses to other buses, the Bus Coupler also does address mapping. It contains four registers which the processor can load with bits to be used for the high order part of a memory address. The processors have sixteen address bits and can thus address 64k bytes. The Bus Coupler registers provide a means for forming 20 bit addresses and thus for reaching 1024k bytes of memory. A diagram of a prototype Pluribus system is shown in figure A-4.

Task scheduling is handled by means of a special hardware device called a Pseudo Interrupt Device (PID). The PID maintains a list of pending tasks (entered by processors or I/O devices). When a processor is ready for a new task, it reads the PID which returns to the processor the identification of the highest priority pending task. Thus task scheduling is done by means of a passive hardware queue, avoiding the problem of routing interrupts to the proper processor.

Techniques used to obtain a very high degree of reliability are described in Ornstein [Ornstein et al, 1975]. A certain amount of help with reliability is derived from the fact that the system is designed to operate within a communications network which has facilities for identifying system malfunctions and also a Network Control Center to provide recovery assistance. More specific reliability features include duplication of every vital hardware resource, isolation between copies of resources so that a single component failure affects only one copy, derivation of periodic checksums on all code, partitioning of the system into reliability subsystems which can be separately verified and use of "watchdog" timers to verify that subsystems are cycling correctly. All malfunctioning processors are disconnected by turning off their bus couplers.

BIBLIOGRAPHY

Heart, FE, Ornstein, SM, Crowther, WR and Barker, WB, A New Minicomputer/Multiprocessor for the ARPA Network, Proc AFIPS, 1973 National Computer Conference, p 529-537.

Ornstein, SM, Crowther, WR, Kraley, MF, Bressler, RD, Michel, A and Heart, FE, Pluribus–A Reliable Multiprocessor, Proc AFIPS, 1975 National Computer Conference, p 551-559.

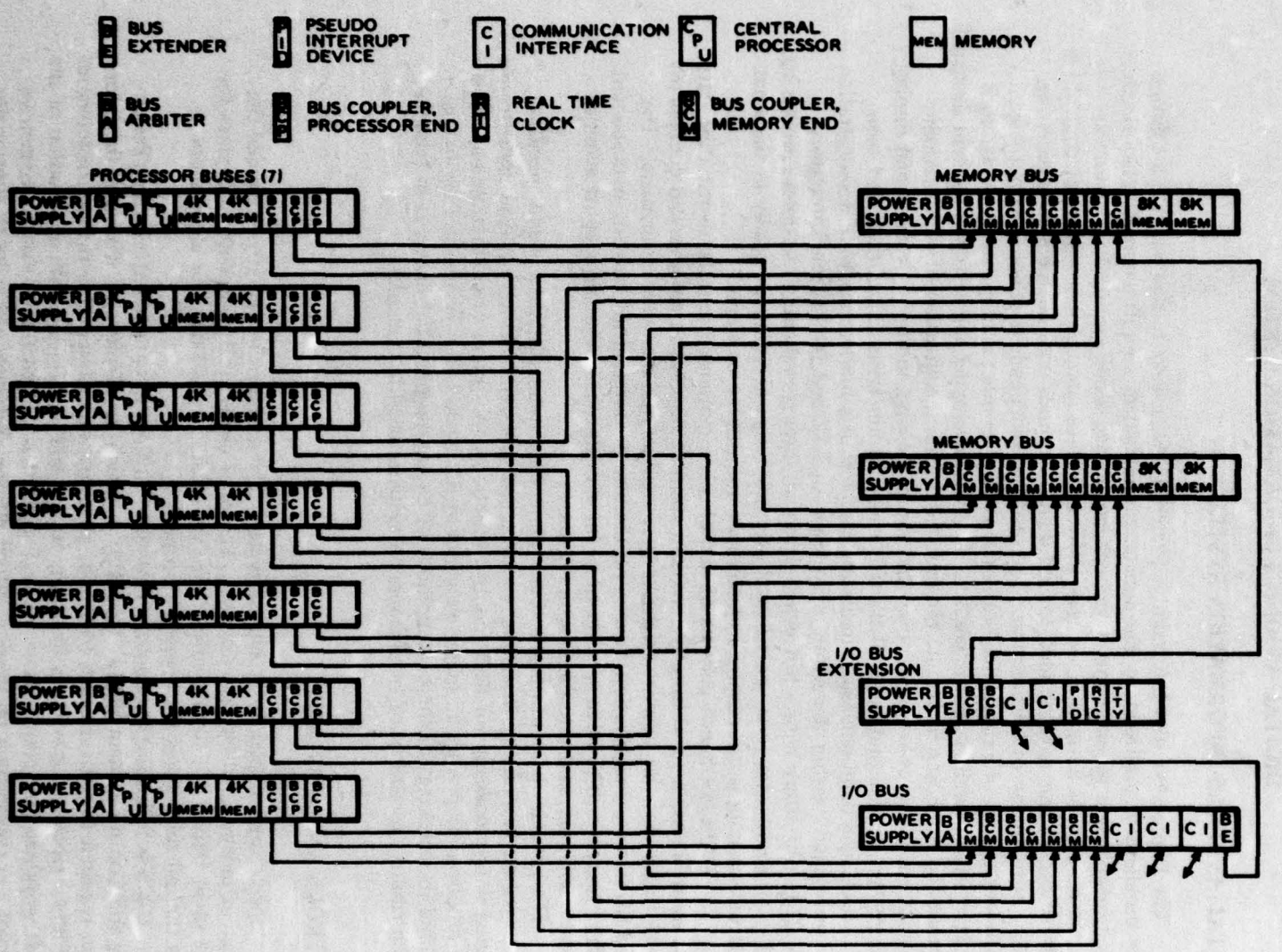


Figure A-4. Prototype Pluribus System.

APPENDIX B

DIGITAL AVIONICS INFORMATION SYSTEM

DIGITAL AVIONICS INFORMATION SYSTEM

The Digital Avionics Information System (DAIS) seeks to demonstrate a solution to the problems of proliferation and nonstandardization of aircraft avionics. Historically, mission information requirements have been established along essentially autonomous subsystem areas such as navigation, weapon delivery, stores management and flight control. Nearly all subsystems have trended toward digital methods with independent processing, transfer, and display of information. The result of independent subsystems put together by different groups of experts has been increasingly complex configurations of avionics with decreasing overall system reliability. DAIS is developing and testing a concept designed to permit the Air Force to assume the initiative in the specification of future avionics configurations. This concept proposes that the processing, multiplex transfer and display functions be common and serve all the other avionic functions on an integrated basis.

DAIS is a comprehensive simulation study with a limited hardware demonstration on a "hot bench" within the Avionics Systems Analysis and Integration Laboratory (AVSAIL). The study with a hot bench demonstration is evaluating multiplex, processing, and display approaches to support logical avionics design decisions especially in the areas of interface standards, processing architectures, and display concepts.

The DAIS hot bench consists of sets of mission-oriented sensors, distribution data buses, an information processing system, and controls/displays in an advanced cockpit for state-of-the-art close air support (night adverse weather) on a simulated airframe. The limited hardware demonstration focuses on the aircraft internal information management, and the program will provide specifications for the advanced close air support avionic hardware, software, and interface requirements.

The program is identifying and recommending standards, criteria and specifications to reduce complexity in avionic systems, and will demonstrate practical performance of digital data processing for flight control functions. A system of compatible models is being produced to predict all major elements of life cycle cost. Techniques are being identified in searching for the best methods of exploiting potential increases in reliability, maintainability, versatility of future weapon systems and life cycle cost.

CONTROLS AND DISPLAYS

Increasing complexity in avionic systems and weapons has greatly expanded the number of independent cockpit displays. The pilot needs most independent displays for only a small fraction of total mission time, yet must scan and monitor all of them to insure that all subsystems are operating properly.

DAIS is following the concept developed in the "Integrated Information Presentation and Control System Study" (IIPACS) in which the displays present only the information required to perform each particular phase of the mission, plus any out-of-tolerance conditions. However, maximum flexibility is achieved by offering both automation of the aircraft workload and pilot options where judgment is needed for mission contingencies. The pilot can concentrate on mission-critical information without having to continually scan a complex array of displays. Loss of a display unit does not wipe out any information presentation, since display functions are shared.

As an alternative to the cockpit complexity brought about by proliferating autonomous subsystems with individually dedicated displays, DAIS is evaluating a cockpit of controls and displays integrated with the overall avionics system. In this evaluation, the latest technologies are examined along with their information transfer requirements, signal structures, and human engineering considerations, for advanced close air support aircraft missions.

DAIS is also studying various implementation approaches for an advanced digital flight control system with high reliability, fast reaction speed, and simplified reprogramming provision. A primary design objective is reduced pilot workload covering a number of missions.

Relief of pilot workload (especially during critical mission phases) and simplicity of shared-display cockpit presentation, together with potential cost, maintenance and retrofit advantages, make the controls and display area a major element in the integrated avionics system. The DAIS computer evaluation includes a cockpit with actual controls and displays wired to the other elements of the system in hardware or simulation models, and to the airframe and environment simulation models, with the cockpit rigged for dynamic and realistic flight simulation. Man-machine interfaces are also evaluated in the realistic flight environment, using outside visual cues and motion cues related to the missions.

Control and display equipment for DAIS includes heads-up display, horizontal situation display, vertical situation display, multifunction display, mass memory, programmable signal generator, mixing storage and synchronization equipment, control panels, multifunction keyboards, and potential integration of the helmet mounted display.

PROCESSORS

The DAIS processor core element consists of a network of identical federated processors with core memory. One processor is programmed to control the multiplex bus traffic interconnecting the sensors to the core elements.

The DAIS multiple processor concept is not dedicated to a particular processor hardware technology, since the elements are identical in form and operation. New hardware technology can be integrated into the system without modifying the design.

Expensive rewiring is eliminated from potential avionic system updates because a standard multiplex bus distribution system is used, and because desired changes in system operation can be made by processor software changes. If either the threat or the mission changes, new or improved sensors can be added easily and inexpensively, through this programming flexibility and the "plug in" acceptance capability of the multiplex data bus.

Mean time between avionics failures, as well as avionics redundancy, and mission effectiveness are all potentially improved by the federated processing system. If one processor fails, the remaining elements can continue to perform the high priority functions with lower priority functions performed in a degraded mode. If several elements fail, the remaining ones will still continue to perform the most important mission functions, since the controlling processor module can reassign the functions to be performed by each processor. Flight system degradation is therefore controlled. With the complete system operating, dynamic range should be improved over single processor systems because of advantages in parallel and simultaneous operation of several processors. Additional protection is gained from decentralization of processors capable of redundant operation, making the function less threat-sensitive.

MULTIPLEX DATA BUS

The heart of DAIS is a time division multiplex data bus system to distribute information through the aircraft in a common format. The common data format for transmitting information—among sensors, processing system, and presentation and control system—is a vital element in overall flexibility since it facilitates the addition of new or different sensors. New sensors can be plugged into the data bus in the same way that a computer terminal is plugged into a time-shared computer network using a telephone line as the data bus. As required, data interface functions will connect sensors to the bus. Thus, the information distribution system operates in the same way regardless of later changes in sensors at the point of information capture. The data bus "sees" new sensors in the same way as the old ones, and communicates in the same way.

The single multiplex bus (with redundant backups) thus replaces a maze of hard wiring with its attendant retrofit problems. The DAIS multiplex bus will handle power controls and stores management as well as avionics functions, and will conform to DoD Multiplex standards (dual redundant for avionics, quad redundant for flight controls). The data bus will be controlled by the DAIS flight processor system and will have standardized multiplex terminal units and modular subsystem interface units.

Almost all electrical signals in the avionics system will be carried on the data bus in the DAIS multiplexing scheme, except for certain high frequency signals such as audio and video, which will be separately wired.

Although the data transfer capability of the final multiplex bus in a production system will be a finite limitation on continuously adding functions or increasing performance, the bus is designed witn ample capacity to meet postulated needs for the foreseeable life of the DAIS aircraft.

SOFTWARE

Although current Air Force software expenditures exceed computer hardware expenditures, there has been relatively little effort in development of software technologies. Software is a critical element in a weapon system and its problems can cause either subsystem or mission failure. Software is also usually the constraining factor in large scale systems development. In some cases, hardware management concepts and hardware techniques have compounded the problems of software management. The DAIS effort is based on the recognition that software design is a vital effort and a prime integrator for the avionics system, and is a key function which has tremendous potential impact on life cycle costs.

In order to achieve DAIS goals, it was necessary to consider software packages at an early design stage along with potential hardware characteristics, as a total system. This permitted hardware/software trade-offs from the standpoint of the avionics system rather than leaving software to correct hardware limitations. (In some cases, the software utilization actually dictates the hardware design standards.) It is also necessary to achieve some basic software architecture and coding standards so that modular software packages can be produced to minimize reprogramming.

DAIS software falls into two categories: support software to aid in the design, development, verification and evaluation of DAIS mission hardware and software; and the DAIS mission software. The support software will also be useful as a basic tool behind field maintenance.

SUPPORT

An automated system for Software Design and Verification (SDVS) makes extensive use of a computer to reduce the design effort, and includes a JOVIAL compiler for the DAIS Mission Software. This compiler will generate programs for both the flight processors and the AVSAIL Computer, a DAIS mission software assembler for the flight processors, an Interpretive Computer Simulator (ICS) of the flight processors, and an External Environment Simulator for interfacing and exercising the ICS. Also included are a microinstruction assembler for the DAIS flight processors and simulations of data bus and sensors.

The automated design system also manages computer language translations, controls the interface with users for maximum ease of communication, and aids in loading required data for analysis. An interactive data base management subsystem is provided for documentation, configuration management, and management control of all DAIS software, including file management facilities and control of simulator facilities. The software design and verification system is generalized and could be applied to the development or maintenance of any Operational Flight Program or easily tailored to other avionic systems.

A Stimulation, Monitor, and Evaluation System, another part of DAIS support software, presents a real-time environment and test scenarios to the DAIS hardware and software under test, and monitors, controls and records the real-time testing. It is capable of presenting selected data in real-time for quick-look evaluations, and of checking out individual functions of the hot bench hardware test. This system also automatically analyzes the recorded data to assess the performance of the hot bench.

MISSION

The DAIS Mission Software corresponds to an Operational Flight Program (OFP) and includes such functions as executive and bus control, navigation, air-to-air and air-to-ground weapon delivery, stores management, electrical power management, control and display processing, integrated test, preflight and postflight checkout, communications management, etc. This software is designed with a basic executive bus control for management and with the ability to vary other software packages or modules, so that applications programs (computations for target acquisition, weapon delivery, etc) may be changed. The system is therefore capable of being easily and quickly reconfigured to accommodate varying mission requirements as well as different sensors.

Maximum use of high order language coding for the Mission Software is required to ease production efforts and to have maximum cost-effective impact on future maintenance, as well as retrofitting. Similarly, a highly modular architectural structure in the mission software will insure that mission-to-mission reconfiguration and hardware changes cause minimal reprogramming. Coding is according to a new set of programming standards designed to produce high quality, well documented, and highly reliable software.

The design of the mission software is for portable modules with maximum flexibility and responsive characteristics. Maintainability and production standards are also desired, with agreement on some basic conventions for input/output, instructions, labeling, etc. These factors should also dramatically lower software acquisition costs. Initial design of the mission software is for implementing the baseline DAIS hardware configuration of federated processors, although the software could be easily converted to other processing concepts.

STANDARDIZATION AND COMMONALITY

Past avionics procurement has produced autonomous subsystems that are often incompatible until a prime contractor adds special interfaces for compatibility. This still results in considerable hardware redundancy without the benefits of redundancy. For example, radar and infrared subsystems may have similar data processors and displays, yet neither can be switched in flight to replace a failed unit. The procedure for obtaining the two items, like other avionic elements, has involved specifications by different organizations and manufacture by different contractors. Thus past procedure did not look for potentially common items and potentially redundant items to see if they could be shared, or to see if complexity could be reduced.

The DAIS premise is that proper system engineering (with a multidiscipline approach) can determine sharability and commonality among such items, and can evaluate the degree of redundancy desirable. The primary object is to be economically efficient with an advanced avionics package responsive to the missions, and to reduce complexity and proliferation in a cost-effective manner. The method is to provide for analysis which plans for integration of the various pieces into an effective avionic system. In contrast to past procedures, feedbacks are required for decisions at various levels during design for integration, rather than integrating the elements for compatibility after individual design efforts and prototype hardware.

When a proposed avionics system is analyzed and common elements are identified (can there be a common data format, common means of data transfer, etc), the foundation is laid for simplicity, sharing, reduced complexity, increased reliability, and reduced cost of ownership. This analysis within a single simulation facility leads to a reservoir of experience which can be applied to a number of different avionics systems, and will lead to increased usage of common avionics in various aircraft. Mass production of common avioric elements will tend to further decrease costs and increase reliability.

Development and acceptance of appropriate standards, especially in software areas, would have a major impact on maintenance. Simplified and integrated design, modular programming and common standards will alleviate the present low utilization rate which results from an inventory of many highly complex and unique systems. Total training costs could be drastically reduced, not only because maintenance personnel are having to become increasingly specialized, but also because (with present complexity) there is a need to carry a sizable minimum personnel level to handle peak periods of load. Also, easily reprogrammable flight computers as DAIS proposes would have important training implications. Each aircraft could become a static flight simulator for on-the-job training, by the addition of the programming module for training.

APPENDIX C

DOCUMENTATION SPECIFICATION FOR DATA PROCESSING ALGORITHM

1.0 INTRODUCTION

1.1 Objective

The objective of this specification is to provide a uniform set of requirements and guidelines for the documentation of signal and data processing algorithms in a standard format.

1.2 Scope

Regarding content, this specification shall present the documenting requirements and guidelines from a systems engineering viewpoint such that computer system architectural implementations and performance criteria can be assessed through quantitative evaluation of computer resources. Documentation shall consist of the following items:

1.2.1 Algorithm description

This item contains a description of an algorithm in terms of operational, functional and mathematical language. It is to be used by system hardware design personnel responsible for design, evaluation and construction of digital processing systems from hardware building blocks of modular functions.

1.2.2 *Performance characteristics*

This item delineates a minimum set of descriptive characteristics of the specified algorithm by which an assessment of its operational performance can be made.

2.0 APPLICABLE DOCUMENTS

2.1 Government documents

None applicable.

2.2 *Non-government documents*

None applicable.

3.0 REQUIREMENTS

3.1 Algorithm description

The following subparagraphs specify a minimum essential content of descriptors by which to document the functional and operational requirements of signals on data processing algorithms.

3.1.1 Purpose of algorithm

This paragraph shall provide a brief written description of the purpose or usage of the algorithm. This description should contain a general discussion of the environment in which the algorithm will operate. It should indicate the relationships with other system components, such as source of input, sensors, radars, displays, data converters, etc. The role within the system assigned to the algorithm should be stated to delineate the functions it is to accomplish.

3.1.2 Computational procedure statement

This paragraph shall provide a detailed description of the processing procedure which will produce a sequence of operations for solving a predefined algorithmic process. This statement shall provide a textual, mathematical, and logical description of the computational processing requirements for each functional operation involved. It shall include the mathematical defining equations for the processing variables. Expression of the computational procedure shall be in terms of a design or programming language such that each step of the algorithm can be precisely defined; the operations to be carried out must be unambiguously specified for each step. These operations which constitute each computational functional description of a data processing algorithm should be expressed such that they can be related to processor architecture, to basic memory storage and to processing execution timing measurements, ie, instruction mix and unit execution times (memory cycle times).

3.1.3 Computational flow chart

A detailed flow chart shall be included which shall depict the processing flow of computational operations to be performed. It shall contain the mathematical equations to be solved. These equations are provided as a basis for calculating the number and type of processing operations to be executed per iteration and as a basis for partitioning to various levels for design and implementation. An example of a computational flow chart is illustrated in figure C-1.

3.1.4 Structured programming flow chart

A detailed flow chart shall be included which shall depict the sequence of commands of the computational operations to be performed and indicate the "flow of control" from command to command. It shall consist of a directed graph of connected symbols which indicates a hierarchical organization of computational procedural operations to be performed with the flow of control indicated as a straight line sequence of commands flowing from top to bottom or from beginning to end. A linear flow of control is indicated by employing one or a combination of three control structures which include a simple single command sequence, a decision or selection sequence which is characterized by a single entry and a converging common or single exit point, or a repetition sequence exemplified by a looping construct. Direct transfers of control either forward or backward within the processing sequence are not used. An example of a structured programming flow chart is illustrated in figure C-2.

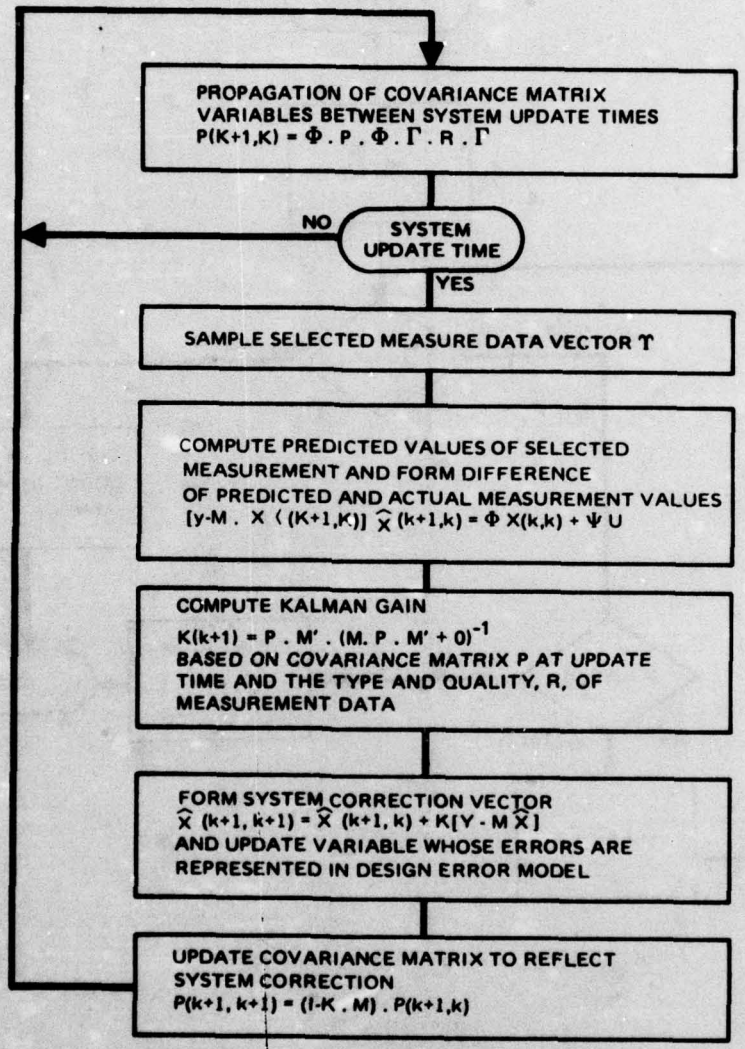


Figure C-1. Sample computational flow diagram.

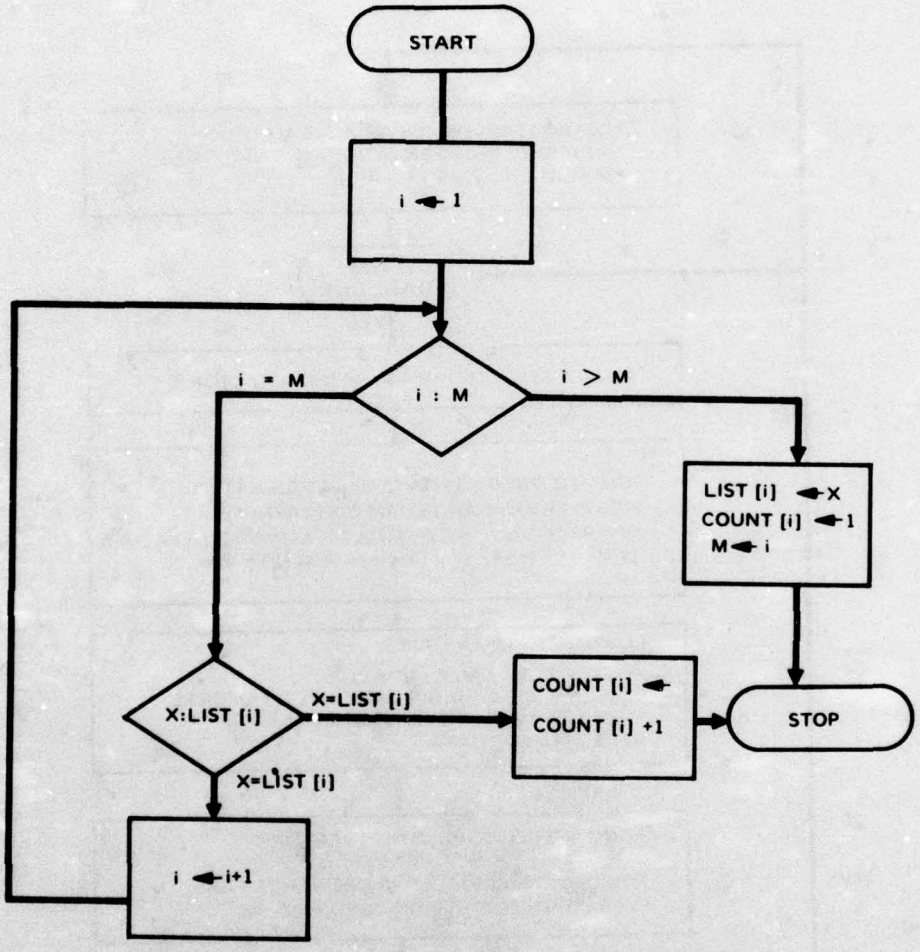


Figure C-2. Sample structured programming flow chart.

3.1.5 Inputs

This paragraph shall provide a description of all input data, including data derived from other functions and from sources external to the processor, such as radars, sonar and other systems. Source of the input, variable identity and components, data format, timing and/or sequencing input data, and associated accuracy limits shall be specified.

3.1.6 Outputs

This paragraph shall provide a description of all output data. This description shall specify the variable identity and components, data format, timing, and/or sequencing requirements, if applicable, and associated accuracy limits.

3.1.7 Memory storage requirements

This paragraph shall provide a description of the criteria used and/or parameterization employed in determining memory storage estimates for designated signal and data processing algorithms. The storage requirements shall be subdivided into two categories of program storage for computer instructions and data storage for both permanent and temporary data types. Presentation of this description shall include the following subparagraphs.

3.1.7.1 Data Storage

This presentation shall relate the data storage requirements to the dimensions of the inputs defined in 3.1.5 and to the number of system state variables which reflect the order of the system being processed. Storage for those data constants and data variables computed outside the algorithm but which require storage and algorithm access shall be included in the storage space requirements. The presentation of data storage requirements may be made in tabular form. An example of such tabular presentation of data storage requirements is illustrated in figure C-3.

3.1.7.2 Program storage

This presentation shall relate the storage requirement for computer instruction storage to those characteristic computational functions performed with the algorithm. These characteristic functions can be further described in terms of a basic computer instruction list which occupies storage units of common size even among processors with differing architecture. The number of each type of instruction and thus storage space occupied can be determined by inspection of computer language subroutines for these functional operations.

3.2 Performance description

The following subparagraphs specify a minimum essential content of descriptions by which to document the characteristic operational performance of a signal or data processing algorithm. The performance description shall be subdivided into three categories of operational characteristics: that which defines the frequency of operation, that which defines the duration of operation, and that which describes the resultant limits of accuracy achieved.

TABLE XXX. DATA VARIABLES AND STORAGE REQUIREMENTS

Variable	Dimension	Storage	Definition
$\hat{X}$	n X 1	2n*	System error estimate vector
P	n X n	$2n^{2*}$	Covariance matrix of error in X
Φ	n X n	n^{2**}	State transition matrix
Γ	n X s	ns**	System disturbance distribution matrix
R	s X s	s^2	System disturbance covariance matrix
M	r X n	nr	Measurement matrix
H	r X r	r^2	Measurement noise covariance matrix
K	n X r	nr	Kalman gain matrix
Z	r X 1	r	Measurement vector of observations
Ψ	n X 1	n*	Control distribution vector
u	n X 1	n	Control vector

Dimension parameters:

n = number of state vector coordinates

r = number of measurement vector coordinates

s = number of disturbances in error model

* Storage required for X (k, k), X (k+1, k), P (k, k) and P (k+1, k) where (k+1, k) notation represents values at t_{k+1} given measurement at t_k

** Data computed outside this algorithm program but which require storage.

Figure C-3. Sample data storage requirements table.

3.2.1 Iteration rate

This paragraph shall describe the computational speed requirements of the algorithm in terms of the necessary frequency of computation. This frequency of computation may be expressed as the number of times each component computational function appears in the total execution path of the algorithm in addition to the number of times per second that each function must be computed. This description shall relate the total iteration requirements to the timing, resolution, and accuracy requirements for each algorithm application environment or to each external component interface of the algorithm. It shall include a definition of the error criteria or resolution accuracy requirements, when applicable, by which iteration termination is determined.

3.2.2 Execution timing

This paragraph shall provide a description of the estimated duration of time required to execute the algorithmic sequence of computational operations on a computer processor. This presentation shall relate the execution timing of component computational functions within the algorithm to the composition of a basic instruction mix required to implement each computational function, and to the frequency of occurrence of each function within the algorithm. The composite execution timing is expressed in terms of the total number of instructions to be executed per unit time. Unit time represents the basic cycle time (memory access time).

In general, instruction lists for different computer processors reflect differing architectures. However, it is reasonable to assume that it is possible to express the execution time for computer instructions in terms of an integer multiple of a basic unit time (microsecond cycle time). Unit time can then be an input parameter for a given computer upon which the particular algorithm implementation is being assessed.

An instruction set can generally be interpreted and classified into four basic categories or classes, with the number of unit times required for execution used as a criterion. These instruction categories include the single unit time subset (eg, branching, index operations, logic operations); double unit time subset (ie, addition, subtraction, store, load, return jump or subroutine jump); multiply instruction with execution time, say t_m, expressed in unit times; and the divide instruction with execution time, t_d, expressed in unit times.

Composite execution timing measures can thus be estimated by determining the number of operations required per instruction class and multiplying by the appropriate multiple of unit time.

3.2.3 Computational accuracies

This paragraph shall provide a description of the computational resultant data resolution and accuracy limitations. This presentation shall relate these accuracy limitations to iteration procedures employed in computation, to computer word size limitations or data format representation limitations.

It shall identify and relate to the processing flow diagram those computational procedural operations where precision and accuracy limitations are critical if reasonable or meaningful resultant resolutions are to be obtained. It should clarify whether or not the resultant resolution is the result of gross approximation methodology or rather of design limitation or constraint upon input and output data.